SBC-2410X 使用手册

Version0.9

广州友善之臂科技有限公司

http://www.arm9.com.cn

http://www.arm9.net

二零零四年十月

(请拿到板子后请务必先阅读说明书，谨防随意破坏系统启动程序 **bootloader(vivi)**

修正

2. 第 145 页：添加 ld.so.conf 内容(2004-11-1)
1. 页眉的“祝您一臂之力”改为“助您一臂之力”(2004-11-1)

前言

SBC-2410X 是由广州友善之臂科技有限公司设计生产的一款基于 ARM9 的嵌入式电脑平台，它基于三星公司的 ARM 处理器 S3C2410X，采用 6 层板设计。S3C2410X 使用 ARM920T 核，内部带有全性能的 MMU(内存处理单元)，它适用于设计移动手持设备类产品，具有高性能、低功耗、接口丰富和体积小等优良特性。SBC-2410X 正是基于此芯片本身的各种特点而设计的。

SBC-2410X 的设计遵循了 S3C2410X 嵌入式芯片的特点，其设计理念参考了当前市场上众多的开发板及嵌入式单板机的优点，并融入最新掌上电脑/手持设备的特点，因此是一款单板机和开发板两用的嵌入式电脑平台。

侧面复位按键的设计位置恰到好处，不会阻挡用户罩在板上的面板；板载采用高质量进口按键，手感特别舒适，经久耐用，而且用户可以随心所欲设计自己的键盘外观，而不需要修改驱动程序；精致的 1220 型板载备份电池更可以保持时间随时随地准确无误；20 针的标准 JTAG 接口让您可以方便的连接各种仿真器；SD 卡、各种优盘移动硬盘、音频输入输出、串口、RJ-45 网线等即插即用；定位孔的设计参考了市面上大量的液晶屏模块尺寸，用户可以使用现成的或者设计自己的液晶模块，扣在板上，完全就是一个标准的 PDA。你可以使用套件提供的各种开发工具按照自己的意图或需要设计各种各样的应用程序，而我们的网站也提供了很多精彩的应用制作实例供您参考，这些实例均有映象文件可以下载演示；整个板子的尺寸只有 120mmx90mm，这仅相当于一个普通 PDA 的大小。

基于以往我们的产品反馈意见，SBC-2410X 进行了严格的电磁，温度，高压脉冲，老化，灰尘等测试，性能稳定，可代替部分工控单板机。在此也十分感谢众多老客户提供的真诚意见！

在软件上，我们首选韩国 MIZI 公司所公布的开放源代码的免费嵌入式操作系统 arm-Linux，基于该平台及其开发工具包，我们编写和移植了丰富的软件供用户使用和参考，如控制台模式下的 Mp3 播放器，Web 服务器(支持 CGI)， Ftp 及其服务器，Telnet 及其服务器，键盘标准驱动程序等等。这些软件的二进制可执行程序及其源代码均收录在所附带的光盘中，并预装在 SBC-2410X 上，用户开机即可使用。

另外，SBC-2410X 也可以运行 WindowsCE4.2.net。WindowsCE 系统包括 Word 阅读器，Excel 阅读器，Power Point 阅读器，图片浏览器，Pdf 阅读器，Word 编辑器，IE 浏览器，媒体播放器，还有一些游戏等。

为让用户能够更加方便的使用 SBC-2410X，我们编写了十分详细的使用手册，它包含在 SBC-2410X 套件中，另外我们会根据用户的要求和反馈意见，将会陆续

在网站上推出一些使用 SBC-2410X 实例以及技术文章给大家。

本手册是 SBC-2410X 的详细使用及开发手册。

最后，欢迎您使用 SBC-2410X，并提出宝贵意见。

编者 capbily/杨斌
capbily@hotmail.com
广州友善之臂科技有限公司
地址：广州市天河路龙口西路龙苑大厦 A1 座 1505 室
电话：020-87530209
邮编：560630
Email：capbily@hotmail.com
网址：http://www.arm9.net
http://www.arm9.com.cn

目 录

第一节 手册指南

1.1 如何使用该手册

下面列出了本手册每个章节的主题：

第一节 指导您如何使用本手册
第二节 MATRIV IV 简介
第三节 详细描述了 SBC-2410X 电路硬件系统的各模块
第四节 详细向您展示如何安装和使用 SBC-2410X
第五节 详细描述了 arm-linux 开发环境的建立
第六节 讲述了如何配置和编译内核以及制作文件系统等
第七节 介绍了在各种情况下如何重新安装系统

1.2 相关连接

1.2.1 友善之臂嵌入式科技有限公司

请浏览我们的网站以得到关于我们更详细的信息。

http://www.arm9.net
http://www.arm9.com.cn

1.2.2 主要外围芯片半导体相关公司

①CS8900A
由Cirrus Logic ISA Ethernet controller chip公司生产制造
http://www.cirrus.com
②MAX3232CSE
由 maxim 公司生产制造
http://www.maxim.com

③**HY57V561620BT**

由韩国现代公司生产制造

http://www.samsungsemi.com

④**K9F5608U0C**

由韩国三星半导体公司生产制造

1.2.3 三星和 mizi 公司

①韩国三星半导体公司

http://www.samsungsemi.com

②韩国 **mizi** 软件公司

http://www.mizi.com

第二节 SBC-2410X 套件概览

本节描述了 SBC-2410X 单板机套件中的各个部件，并且说明了使用时的 PC 机系统需求。

2.1 套件内容

SBC-2410X 套件由包含以下几部分组件：

- 一片 SBC-2410X 单板机(预装 Linux)
- 一张 DVD 光盘
- 一条 DB9 标准直连串口线
- 一条 RJ-45 水晶头对等网线
- 一条 USB 电缆
- 一条 JTAG 电缆
- 一个 5V/2.5A 专用开关电源



图 2-1 SBC-2410X 套件内容

2.2 SBC-2410X 简介

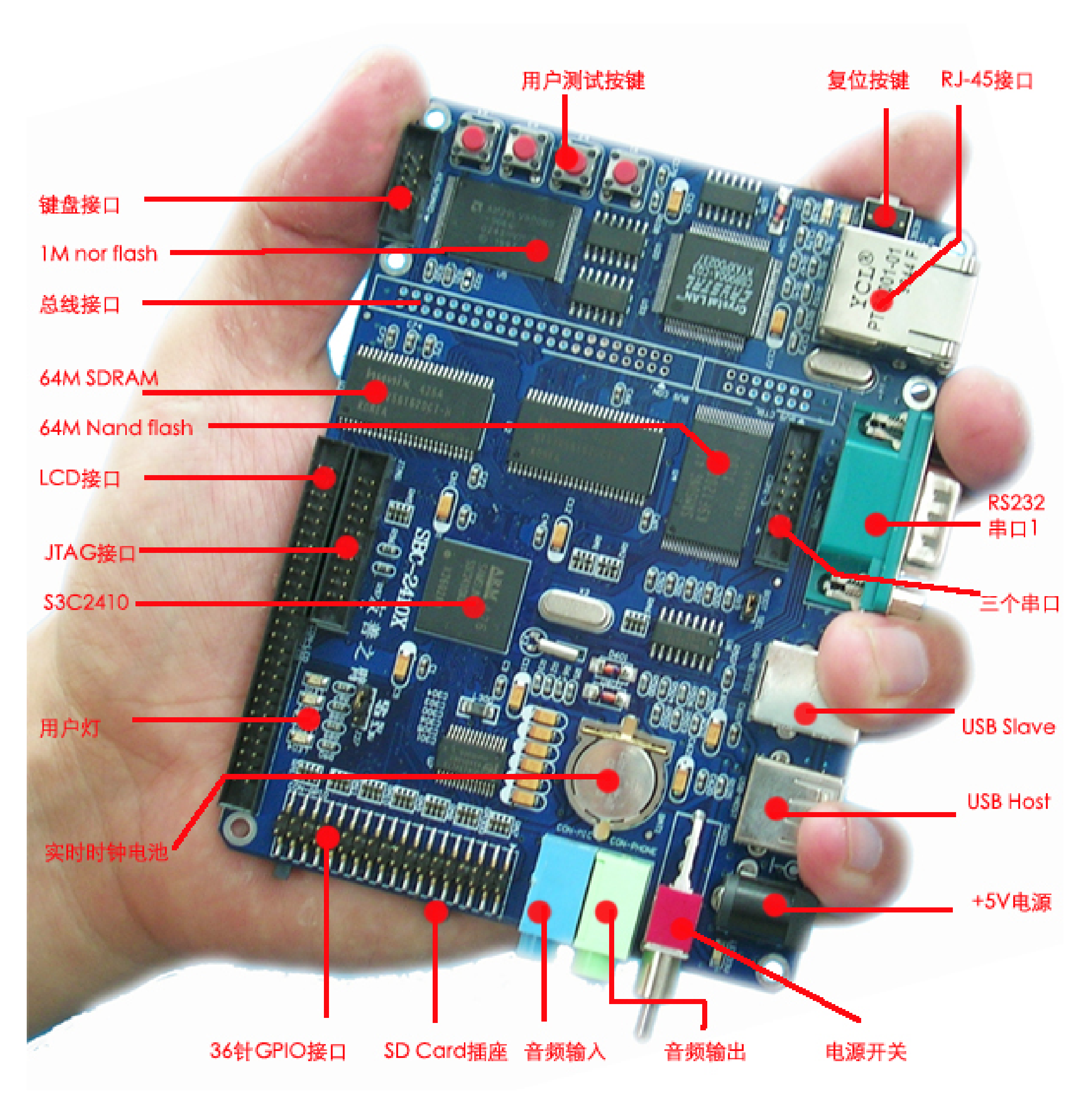


图 2-2 SBC-2410X 概览

SBC-2410X 基于三星公司的 ARM 处理器 S3C2410X，采用 6 层板设计。S3C2410X 使用 ARM920T 核，内部带有全性能的 MMU(内存处理单元)，它适用于设计移动手持设备类产品，具有高性能、低功耗、接口丰富和体积小等优良特性。SBC-2410X 正是基于此芯片本身的各种特点而设计。

SBC-2410X 在尽可能小的板面上(120mmx90mm)集成了 64M SDRAM、64M Nand Flash、1M Boot Flash、RJ-45 网卡、音频输入与输出、USB Host、USB slave、标准串口、SD 卡插座、

用户按键和一些用户灯等设备接口，并且使用 2.0mm 插针槽引出 CPU 的大部分信号引脚，不仅可以作为嵌入式电脑系统的一个主板模块，而且适合于项目或产品的原型设计。

在软件上，我们首选使用韩国 mizi 公司所公布的免费嵌入式 Arm-Linux 操作系统，基于该平台及其开发工具包，我们编写和移植了丰富的软件供用户使用和参考，如 Mp3 播放器，媒体播放器，Web 服务器，Ftp/Telnet 服务器等。这些软件及其大部分源代码均收录在所附带的光盘中，并预装在 SBC-2410X 参考板上，用户开机即可使用。

另外，SBC-2410X 使用了目前 Flash 设备上最优越的 YAFFS 文件系统，提供 62M 可读写空间。

2.2.1 原型及样机设计

本手册的第三节对 SBC-2410X 的各个硬件模块和接口进行了详细的描述，并在所附的光盘中提供了完整的 Protel99SE 格式原理图及封装库；手册中还详细介绍了基于 Linux 平台的开发环境的建立和安装，并且提供了丰富的驱动程序和应用程序示例代码，非常适合于原型及样机的设计和评估，并可以直接作为嵌入式系统主控模块来使用。

2.2.2 移动设备 DIY

另一方面，SBC-2410X 作为开发板，同其开发板最大的区别或许就是，它在尽可能小的板面(尺寸只有 120mmx90mm)上集成了 64M SDRAM、64M Nand Flash、1M Boot Flash 网卡、音频、USB Host、USB slave、标准串口、SD 卡插座、用户按键和一些用户灯等设备接口，并且引出 CPU 的大部分信号引脚，它几乎具备了一台 PC 所有的接口，因此很适合移动设备爱好者 DIY 自己喜爱的东西；如大容量口袋型 Mp3 播放器，车载 Mp3，数码相册等等，如果您不懂得如何做，我们的网站陆续会有大量的 DIY 实例供您参考。即使是作为一名专业做开发的工程师，您也可以拿着自己在实验室里做的“产品”向您的朋友展示您的才华，因为它可以随时装在口袋里。

2.2.3 软硬件性能特点

SBC-2410X 硬件标准特性：

序号	名称	描述
1	CPU	Samsung S3C2410X，200MHz 主频，最高 266Mhz
2	ROM	1 MB AMD Flash ROM 64M Samsung Nand Flash ROM
3	RAM	32Mx2 SDRAM
4	LAN	一个 10M Ethernet，RJ-45 接口
5	Serial	1 个 DB9 标准串口
6	USB	1 个 USB Host A 型接口(USB1.1 协议) 1 个 USB Host B 型接口(USB1.1 协议)
7	Audio	一个音频接口(双声道，可直接接耳塞) 一个音频输入口
8	RTC	外接 32.768KHz 的晶振，带有备份电池，可保持时钟
9	JTAG	一个 20 针(2.0mm 间距)标准的 JTAG 接口，主要用来下载 bootloader
10	SD Card	SD Card 插座
11	LED	4 个可编程用户 LED(带驱动程序)
12	Keypad	4 个可编程用户按键(带有驱动程序)
13	Switch	一个电源开关
14	Reset	一个复位按键
15	POWER	一个开关电源+5V 供电
16	Fixed Hole	6 个定位孔(内径 3mm，外径 5mm)
17	Boardsize	120(L)x90(W)mm

SBC-2410X 专有接口特性：

序号	名称	描述
1	COM1-3	14 针 2.0mm 间距接口直接引出 CPU 内部三串口
2	KEYBOARD	10 针 2.0mm 间距接口引出 IO 键盘
3	CON-LCD	LCD/STN 液晶屏接口(50 针 2.0mm 间距)，可以接各种单色,伪彩,真彩液晶屏，含有触摸屏接口
4	CON-GPIO	GPIO 等接口(36 针 2.0mm 间距)，含有 10 个中断引脚，6 路 AD 输入，1 个 SPI 接口，I2C 接口，2 个时钟输出，2

		个 GPIO 口
5	BUS CON	总线接口(44 针地址数据线,14 针控制线),带有 16 位数据线,25 位地址线等信号

SBC-2410X 软件特性(for Linux)：

名称	功能特性	说明
BIOS	bootloader	启动系统(可以设置启动时间)
	Xmodem	支持 Xmodem 传输协议
	Update Flash	支持更新 Flash
	Set Kernel Parameter	支持设置内核参数
	Set Partation	支持设置分区
	Etc.	其他
内核	Linux kernel 2.4.18	使用 2.4.18 稳定内核
	ROM/CRAM/EXT2/FAT32/NFS/YAFFS file system	支持 ROM/CRAM/EXT2/FAT32/NFS/YAFFS 等文件系统
	etc.	其他
驱动支持	System Interrupt & Timer Driver	系统中断和系统时钟驱动
	Serial device driver	三个串口驱动
	Block memory device driver	块设备驱动
	Flash memory device driver	Nand Flash 驱动
	10Base-T external Ethernet device driver	外接 10M 以太网卡驱动
	RTC (Real Time Clock) Driver	实时时钟驱动
	USB Host driver	USB Host 驱动
	USB Slave	USB Slave 驱动
	LEDS	LEDS 灯驱动
	Buttons	用户按键驱动
	Many Country Language Support	国际化支持
	LCD	液晶(LCD)驱动
	Frame Buffer	Frame Buffer
网络协议及网络应用程序	TCP/IP	完整的 TCP/IP 协议
	NFS	支持 NFS Server 和 NFS Client
	Telnet Server	Telnet 服务器
	File transfer (FTP client / server)	FTP 传输服务(包括登录和服务器)
	Remote login (telnet)	Telnet 远程登录
	Web server (HTTP v1.1, boa)	Webserver(boa)

	Web base management suite (Sample only)	可通过网页管理目标板
系统升级	Console & FTP	在控制台下使用 FTP 升级系统
	NFS	通过网络文件系统升级
	Mobile Storge	通过移动存储升级
配置系统和服务	Local console through RS232 and remote telnet configuration and management(Sample Only)	通过串口控制台或 telnet 远程登录配置系统
	WEB configuration and management(Sample Only)	通过浏览器进行简单配置
	Network ping	使用 ping 检查系统
	Ifconfig, route, inetd	提供 ifocnfig, route,inetd 等网络配置和服务程序
	login, sh, echo, discard	登录和其他程序
基本工具	cat, chmod, discard, echo, flashfsd, flashwrite, free, genhtml, hostname, init, kill, loader, ls, mkdir, mount, ps, reboot, rm, smanaged, sysconf, yes,insmod, lsmod, rmmod	Linux 常用命令
实用应用程序	madplay	控制台下的 mp3 播放器
	leds	控制用户 LED
	buttons	键盘使用程序
图形界面	Qt/ EMBEDED	一个非常成熟的嵌入式图形用户界面系统，带有很多丰富的应用程序
	Microwindows/NanoX	一个轻型的嵌入式图形系统
数据库	armsql	适用于嵌入式系统的一个功能强大的数据库

2.2 光盘介绍

SBC-2410X 所附带的光盘主要包含以下两个目录：

- MIZI 目录，由 mizi 的 ISO 的光盘影像文件提取出来，它可以在 mizi 网站免费下载得到：
  - 应用实例代码
  - 三星评估板数据手册
  - S3C2410X 数据手册

- Mizi 开发包使用手册(部分有韩文)
- 三星评估板电路原理图(ORCAD/PDF 格式)
- GNU gcc 交叉编译工具链

- SBC-2410X 目录，适合于 SBC-2410X 开发使用主要包含以下内容：
  - 本手册的电子文档 (PDF 格式)
  - 使用嵌入式 Linux 开发时所需要的各种程序及文件，请参考“Linux for SBC-2410X”一节
  - 使用 WindowsCE4.2.net 开发时所需要的各种程序及文件，请参考“WindowsCE For SBC-2410X”一节
  - SBC-2410X 原理图(Protel 格式)
  - SBC-2410X 元器件数据手册(pdf 格式)

2.3 系统需求

当使用 SBC-2410X 进行开发时，它必须使用串口、JTAG 电缆和网络接口与 PC 相连接，一台开发用的 PC 推荐使用以下配置：

- 奔腾 2.0G(或更高)或兼容 PC
- 安装 WindowsXP/2003/Redhat 9.0 操作系统
- 256M 内存
- 40GB 硬盘可用空间
- 一个 CD-ROM 光驱
- 一个串口
- 一个并口
- 一个网口
- 一个 USB 接口

注意：建议使用 Intel 芯片组的主板，否则 Jtag 连接会出现不稳定问题。

第三节 快速安装使用指南

Step 1 打开包装



图 **3-1** 打开 **SBC-2410X** 套件包装

Step2 检查套件



图 **3-2** 检查套件内容

Step3 连接 LCD，串口及电源

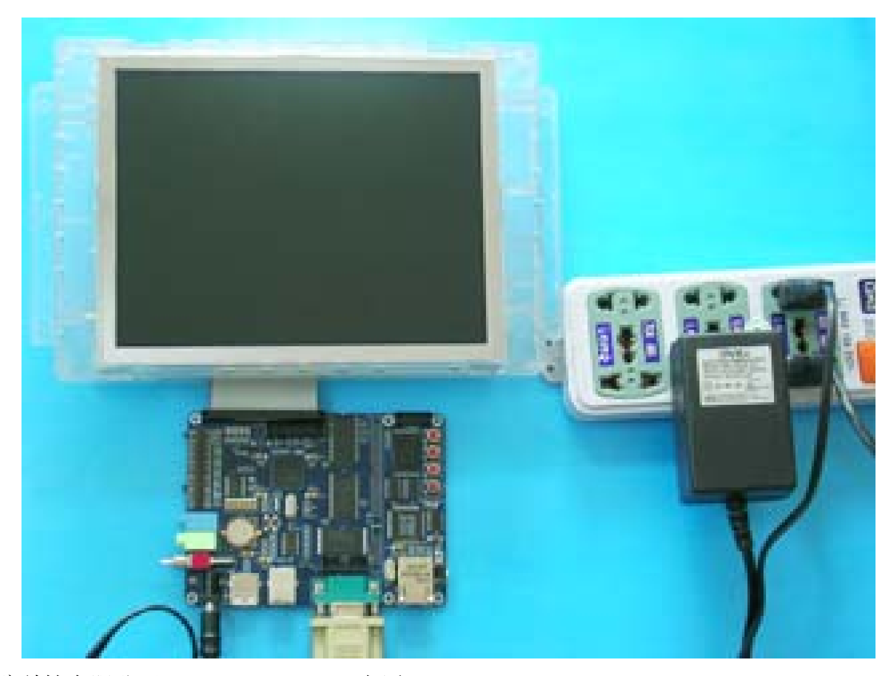


图 **3-3** 连接电源，串口和 **LCD**

Step4 开机画面(预装 Linux)

(1) LCD 显示



图 **3-4** 打开电源，等几秒钟，出现 **Qt** 开机界面

(2) 串口信息

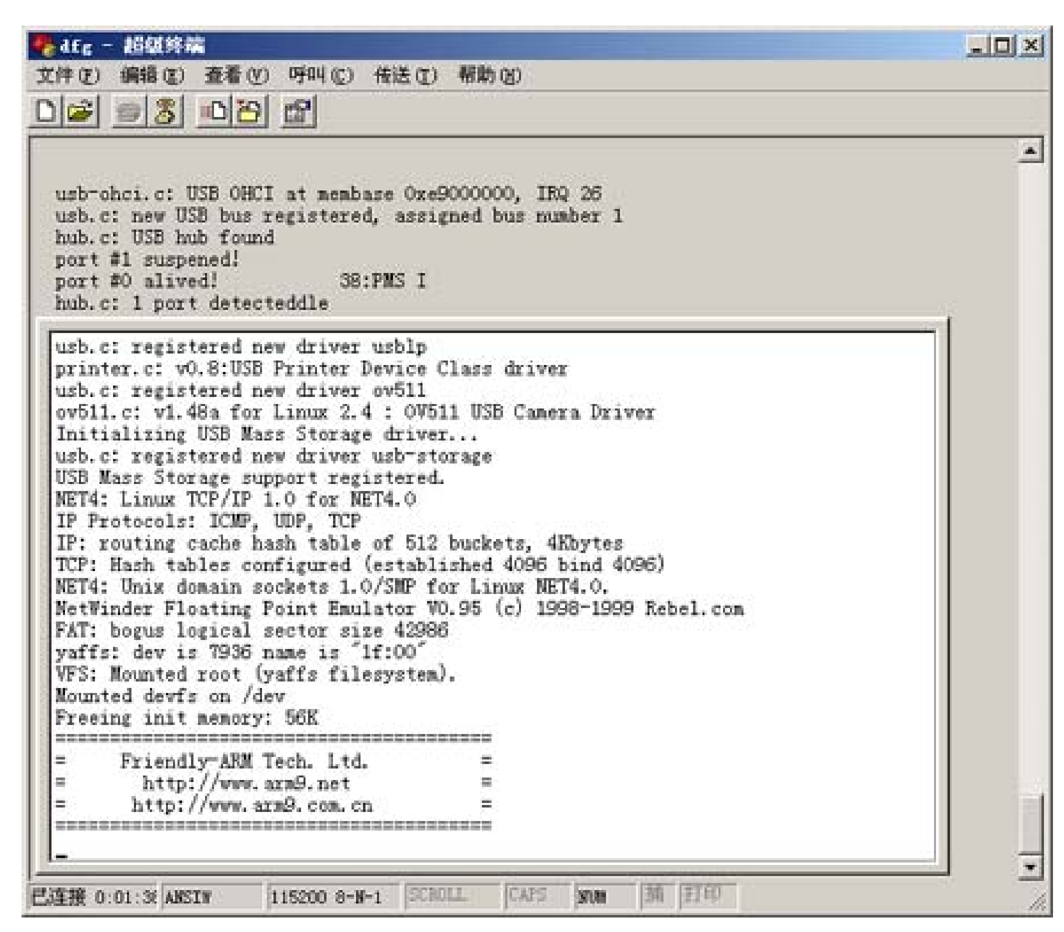


图 3-6 打开电源，预装 **Linux** 串口**(115200, 8N1)**启动信息

Step5 开机画面(预装 WindowsCE)

(1) LCD 显示



图 **3-7** 打开电源开关，等几秒钟，出现 **WindowsCE** 开机画面

(2)串口信息

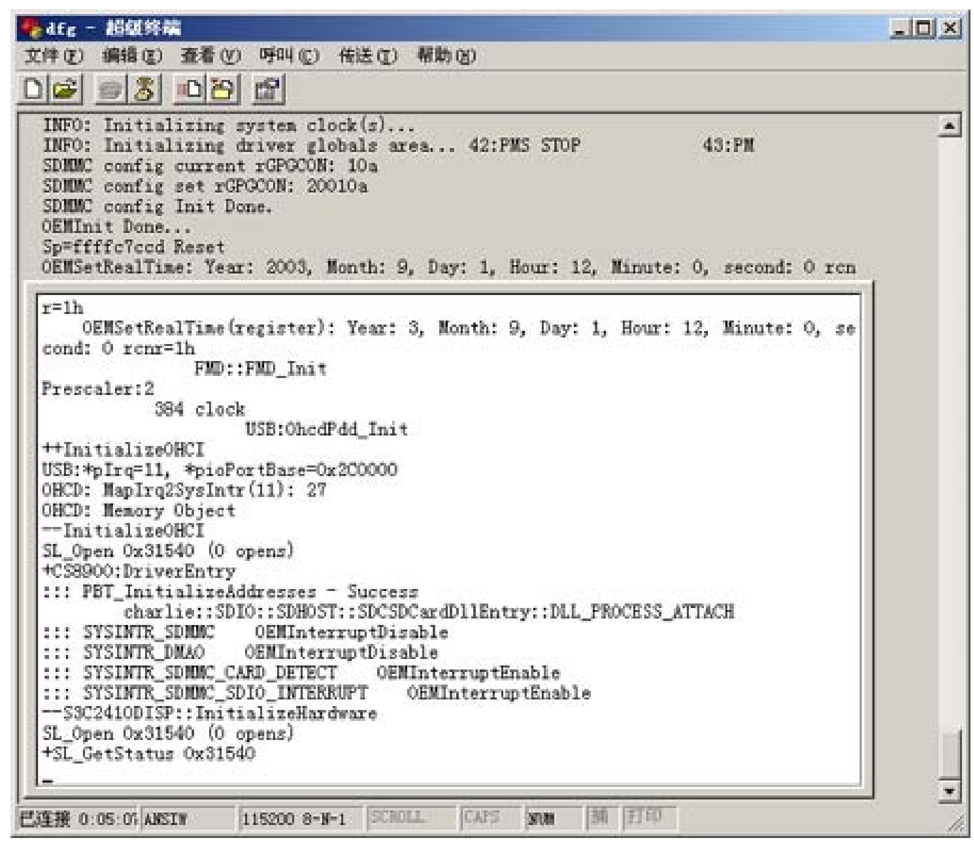


图 **3-8** 图 **3-6** 打开电源，预装 **WindowsCE** 串口**(115200, 8N1)**启动信息

第四节 SBC-2410X 的硬件系统

4.1 SBC-2410X 概览

下图是 SBC-2410X 的图片说明：

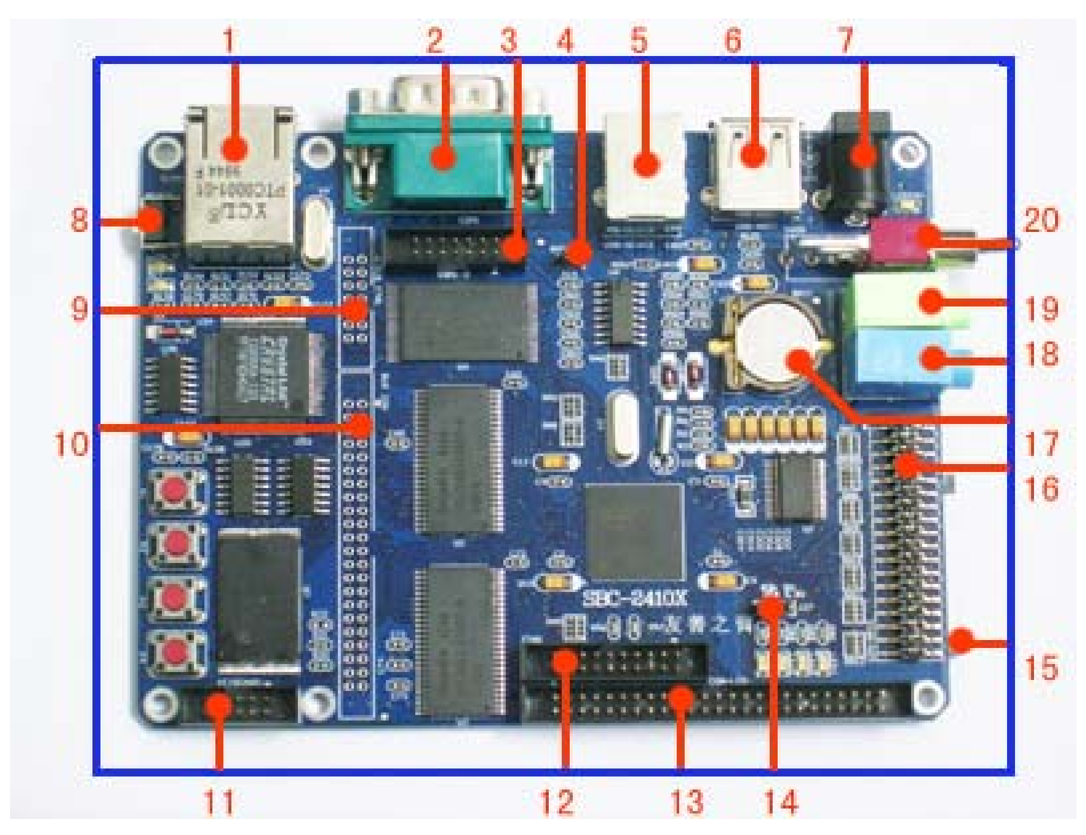


图 3-1 SBC-2410 接口概览

表 3-1：SBC-2410X 接口说明：

序号	对用原理图名称	说明
1	RJ45	10M 以太网接口
2	COM1	RS232 串口 1
3	COM1-3	CPU 直接引出的 COM1,2,3
4	BOOT SEL	启动 ROM 选择
5	USB DEVICE	USB 从设备

6	USB HOST	USB 主设备
7	CN1	+5V 电源插座(内正外负)
8	RESET	复位按钮(位于侧面)
9	BUS CTRL	总线控制信号等
10	BUS CON	系统总线(16 位数据，25 位地址)
11	KEYBOARD	键盘接口
12	JTAG	JTAG 接口
13	CON-LCD	LCD 接口
14	J27	3.3V/5V LCD 电源选择跳线
15	SD Card	SD 卡插座
16	CON-GPIO	通用 IO 口，AD 输入口等
17	BAT1	时钟备份电池插座
18	CON-MIC	单声道音频输入插座
19	CON-PHONE	双声道音频输出插座
20	S600	电源开关

4.2 SBC-2410X 接口介绍

关于跳线设置如下图所示：

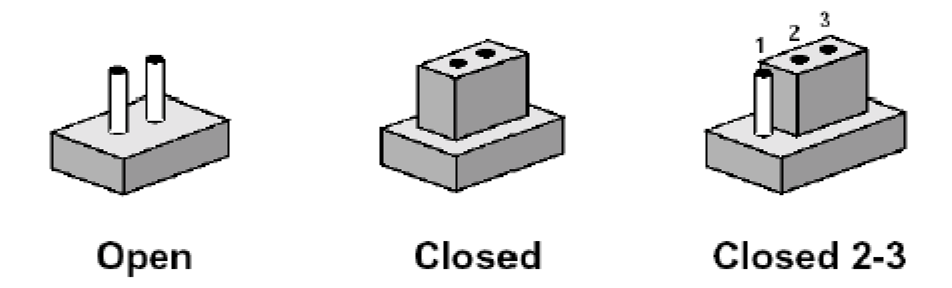


图 3-2 跳线设置示意图

以下详细介绍各个接口的引脚定义等。

4.2.1 LAN：RJ45

SBC-2410X 的网卡插座和外接网线/HUB 引出网线如图 3-3 所示：

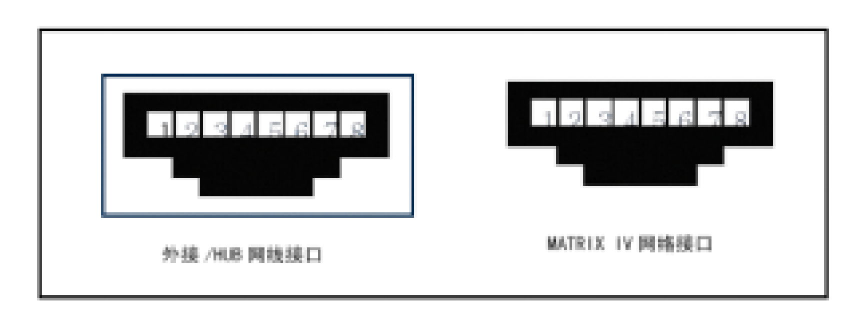


图 3-3 SBC-2410X 网络接口

表 3-2 SBC-2410X 网络接口引脚描述

引脚	名称	描述
1	**TX+**	**Transmit Data+**
2	**TX-**	**Transmit Data-**
3	**RX+**	**Receive Data+**
4	**NC**	**Not connected**
5	**NC**	**Not connected**
6	**RX-**	**Receive Data-**
7	**NC**	**Not connected**
8	**NC**	**Not connected**

4.2.2 串口 1：COM1

Samsung S3C2410X 本身带有 3 个标准串口，SBC-2410X 引出串口 1 并接到 DB9 插座上，为方便用户设计，三个串口通过 14 针间距 2.0mm 插座 COM1-3 直接从 CPU 引出(详见 COM1-3 说明)。

如图 3-3 是 SBC-2410X 串口 1 的信号引脚图：

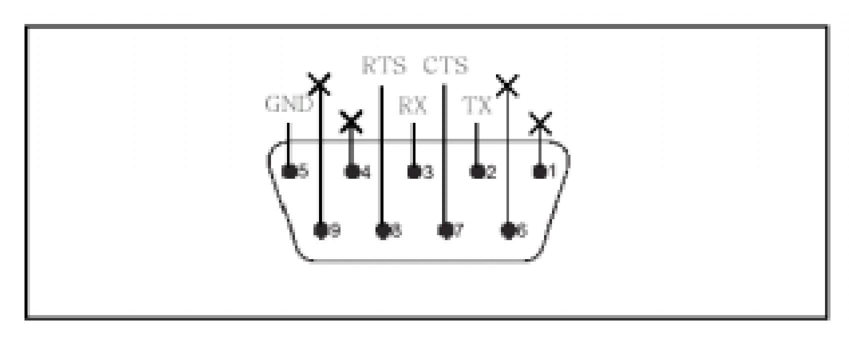


图 3-3 SBC-2410X 上串口 1 的引脚说明

4.2.3 串口 1-3：COM1-3

如图为 SBC-2410X 板上 COM1-3，各引脚说明见表 3-3。

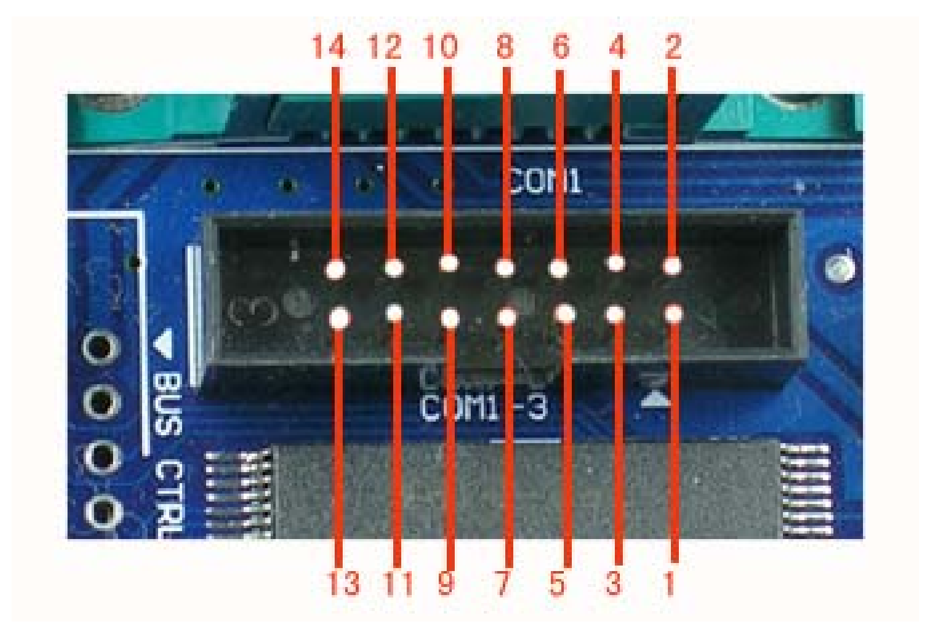


图 3-4 SBC-2410X 之串口引脚 COM1-3

表 3-3：COM1-3 引脚说明

序号	说明	序号	说明
1	VDD33V	2	VDD33V
3	nCTS0	4	nRTS0
5	TXD0	6	RXD0
7	TXD1	8	RXD1
9	TXD2	10	RXD2
11	NC	12	NC
13	GND	14	GND

注：NC 表示没有连接(下同)

4.2.4 ROM 启动选择：BOOT SEL

Samsung S3C2410 支持 Nor Flash 和 Nand Flash 启动，在 SBC-2410X 上可以通过 BOOTSEL 跳线设置启动方式，各见下图说明。



图 3-5 **BOOT SEL** 短接表示选择 **Nand Flash** 启动

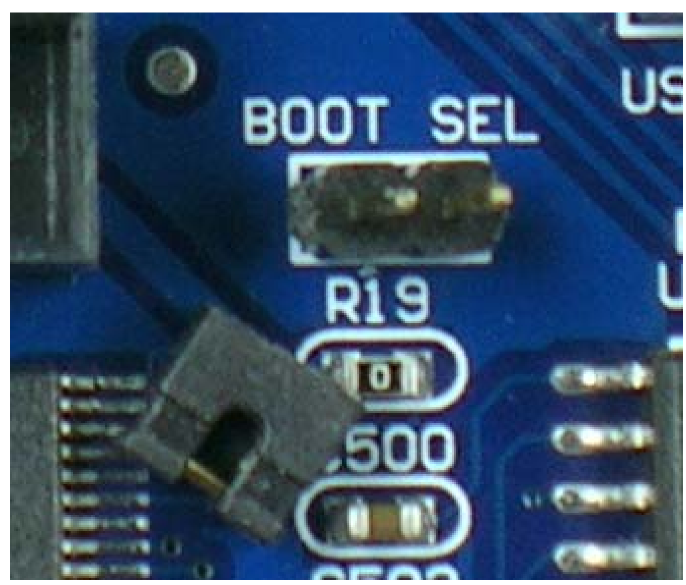


图 3-6 **BOOT SEL** 不短接表示选择从 **Nor Flash** 启动

4.2.5 USB 接口

Samsung S3C2410X CPU 带有 2 个 USB 接口，一个作为 Host(可配置为 Slave 模式)，一个是 Slave。SBC-2410X 把这两个接口均引出，其中 USB HOST 使用 PC 上常见 A 型口，USB DEVICE 使用 B 型口，见下图：

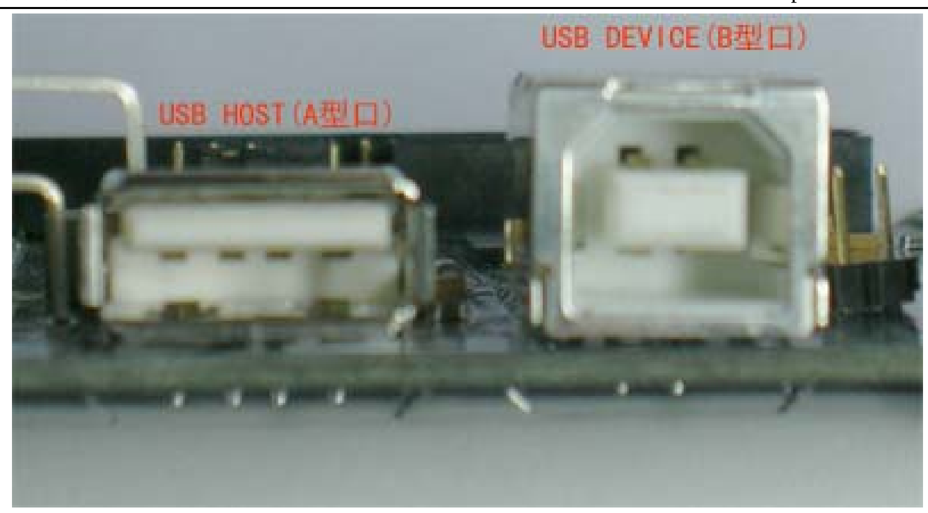


图 **3-7 USB** 主从口



图 **3-8** 使用 **USB HOST** 接移动存储设备

使用 **USB DEVICE** 连接电脑

4.2.6 电源插座和电源开关：CN1 和 S600

SBC-2410X 使用+5V 直流电源供电，并在电源插座的旁边标明了插座的极性，使用电源开关 S600 可以控制板上电源系统的通断，如下图所示：

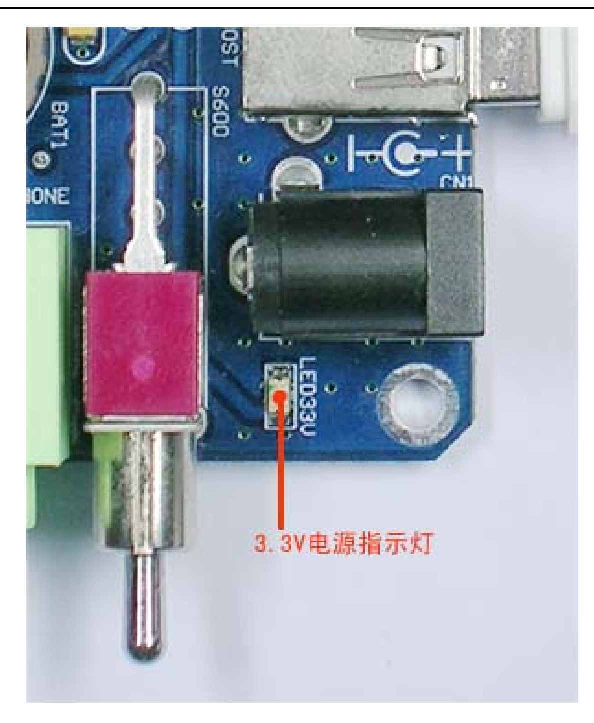
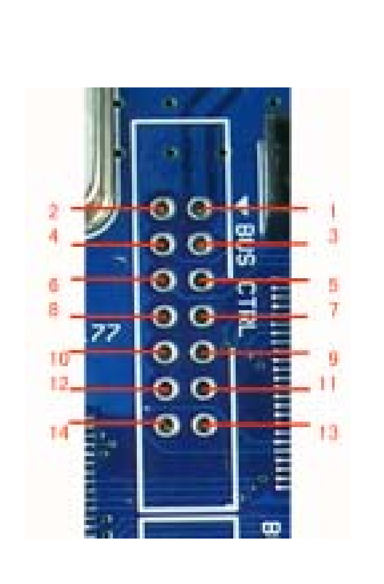


图 3-9 SBC-2410X 的电源插座与开关及 3.3V 电源指示灯

4.2.7 系统总线接口

SBC-2410X 使用两组 2.0mm 间距接口引出 CPU 内部的系统总线。其中 BUS CTRL 接口主要为系统总线控制信号，BUS CON 主要为地址数据线等，分别如下图表所示。



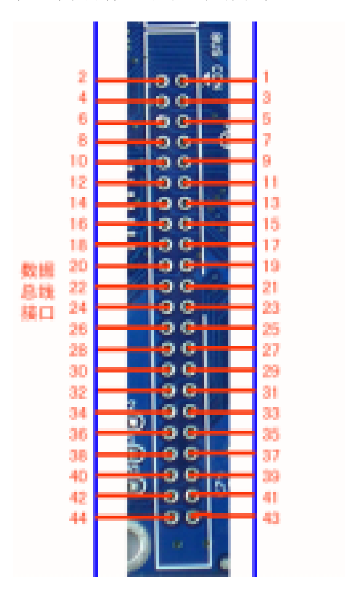


图 **3-10 SBC-2410X** 控制总线和数据总线接口

表 **3-4 SBC-2410X** 控制总线接口**(BUS CTRL)**引脚定义

序号	引脚定义	序号	引脚定义
1	VDD5V	2	VDD33V
3	nRESET	4	nGCS4
5	nGCS2	6	nGCS1
7	LnWBE0	8	LnWBE1
9	LnWBE2	10	LnWBE3
11	EINT3	12	EINT8
13	GND	14	GND

表 **3-5 SBC-2410X** 地址数据总线接口**(BUS CON)**引脚定义

引脚定义	序号	序号	引脚定义
LDATA1	2	1	LDATA0
LDATA3	4	3	LDATA2
LDATA5	6	5	LDATA4
LDATA7	8	7	LDATA6
LDATA9	10	9	LDATA8
LDATA11	12	11	LDATA10
LDATA13	14	13	LDATA12
LDATA15	16	15	LDATA14
LADDR1	18	17	LADDR0
LADDR3	20	19	LADDR2
LADDR5	22	21	LADDR4
LADDR7	24	23	LADDR6
LADDR9	26	25	LADDR8
LADDR11	28	27	LADDR10
LADDR13	30	29	LADDR12
LADDR15	32	31	LADDR14
LADDR17	34	33	LADDR16
LADDR19	36	35	LADDR18
LADDR21	38	37	LADDR20
LADDR23	40	39	LADDR22
LnOE	42	41	LADDR24
nWAIT	44	43	LnWE

4.2.8 音频接口

Samsung S3C2410X 带有 I2S 音频总线，因此 SBC-2410X 使用了一片 I2S 接口的菲利普解码芯片 UDA1341，通过该芯片和相应的软件，用户可以播放 mp3 以及 wav 格式的音频文件，双声道的声音可以从背面的耳塞插孔直接接至耳塞或者音箱，音频输入接口可以录音。

4.2.9 键盘

SBC-2410X 单板机本身带有 4 个 IO 方式的用户可编程按键，同时把这些 IO 口引出到一个 2.0mm 间距的插座上，方便用户自由设计键盘方式。

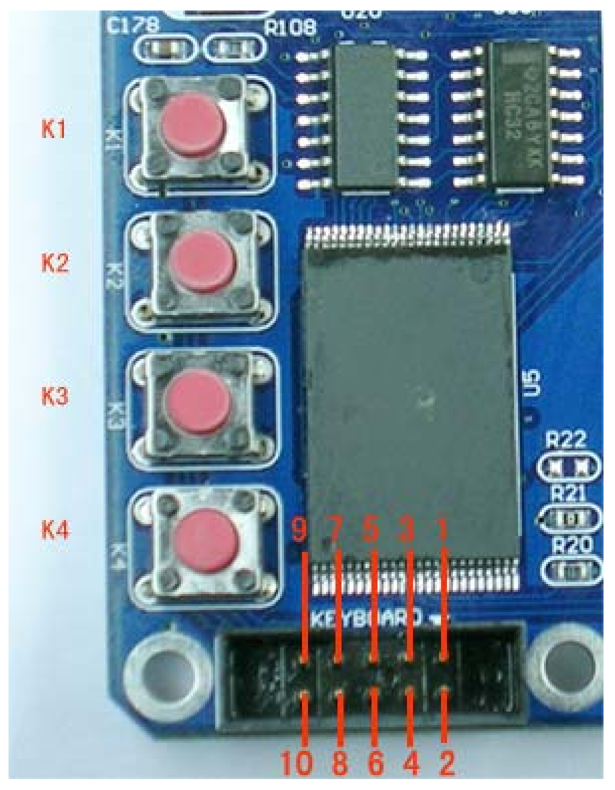


图 3-11 SBC-2410X 按键及按键引脚

表 3-6 SBC-2410X 按键引脚(KEYBOARD)说明

序号	定义	序号	定义
1	VDD33V	2	VDD33V
3	EINT7	4	EINT1
5	NC	6	EINT2
7	NC	8	EINT3

9	GND	10	GND

4.2.10 用户 LED

SBC-2410X 单板机作为一个开发系统，为用户准备了 4 个 IO 方式的用户可编程 LED 灯(绿色)。

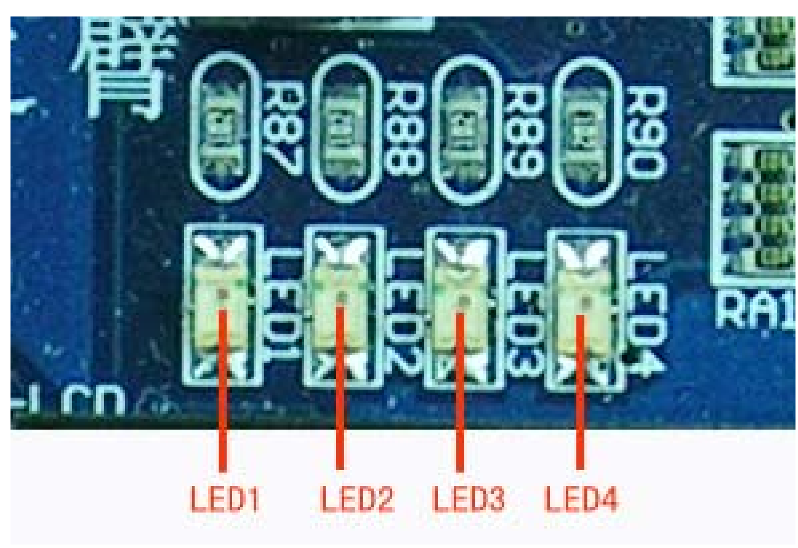


图 3-12 SBC-2410X 用户可编程 LED

它所占用的资源如表 3-7 所示。

表 3-7 用户指示灯占用 CPU 资源列表

序号	名字	CPU 端口复用资源
1	LED1	nXDREQ1/GPB7
2	LED2	nXDREQ0/GPB8
3	LED3	nXDACK1/GPB9
4	LED4	nXDACK0/GPB10

4.2.11 LCD 接口

Samsung S3C2410X 支持以下液晶屏，SBC-2410X 引出了其内部所有液晶控制器引脚信号，如图 3-14 所示，表 3-8 描述了这些脚的定义说明。

STN LCD displays:

— Supports 3 types of LCD panels: 4-bit dual scan, 4-bit single scan, and 8-bit single scan display type

— Supports the monochrome, 4 gray levels, and 16 gray levels

— Supports 256 colors and 4096 colors for color STN LCD panel

— Supports multiple screen size
Typical actual screen size: 640×480, 320×240, 160×160, and others
Maximum virtual screen size is 4Mbytes.
Maximum virtual screen size in 256 color mode: 4096×1024, 2048×2048, 1024×4096, and others

TFT LCD displays:

— Supports 1, 2, 4 or 8-bpp (bit per pixel) palettized color displays for TFT

— Supports 16-bpp non-palettized true-color displays for color TFT

— Supports 24-bpp non-palettized true-color displays for color TFT

— Supports maximum 16M color TFT at 24bit per pixel mode

— Supports multiple screen size
Typical actual screen size: 640×480, 320×240, 160×160, and others
Maximum virtual screen size is 4Mbytes.
Maximum virtual screen size in 64K color mode: 2048×1024 and others

图 3-13 S3C2410X 支持的液晶种类(摘自 S3C2410X 数据手册)

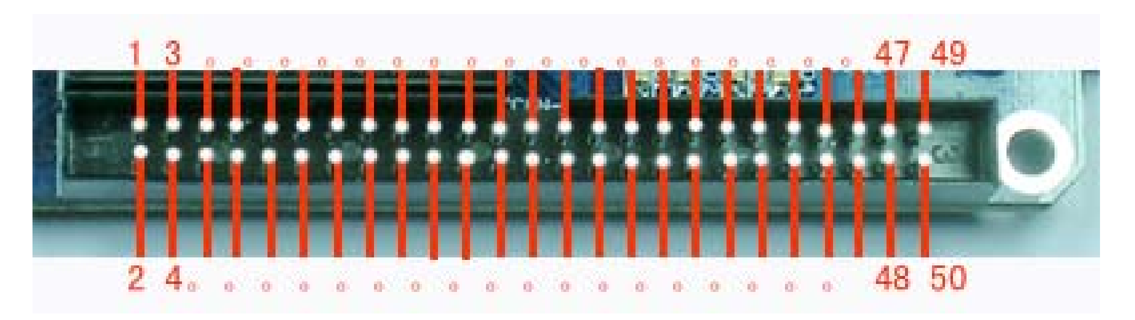


图 3-14 SBC-2410X 板上 50 针 LCD 接口(请注意 1 脚位置)

表 3-8 LCD 接口定义

引脚定义	序号	序号	引脚定义
VCC(注)	2	1	VCC
GND	4	3	VCC
VD0	6	5	nRESET
VD2	8	7	VD1
VD4	10	9	VD3
VD6	12	11	VD5
VD8	14	13	VD7
VD10	16	15	VD9
GND	18	17	VD11
VD13	20	19	VD12
VD15	22	21	VD14

VD17	24	23	VD16
VD19	26	25	VD18
VD21	28	27	VD20
VD23	30	29	VD22
LCD_PWREN	32	31	GND
LCDVF1	34	33	LCDVF2
VM/VDEN	36	35	LCDVF0
VLINE/HSYNC	38	37	VFRAME/VSYNC
LEND	40	39	VCLK/LCD_HCLK
GND	42	41	nDIS_OFF
nXPON	44	43	XMON
GND	46	45	AIN7
nYPON	48	47	YMON
GND	50	49	AIN5

注：通过 J27 跳线可以在 3.3V 和 5V 之间选择 LCD 的供电电压。

1-2 短接选择 5V 供电； 2-3 短接选择 3.3V 供电

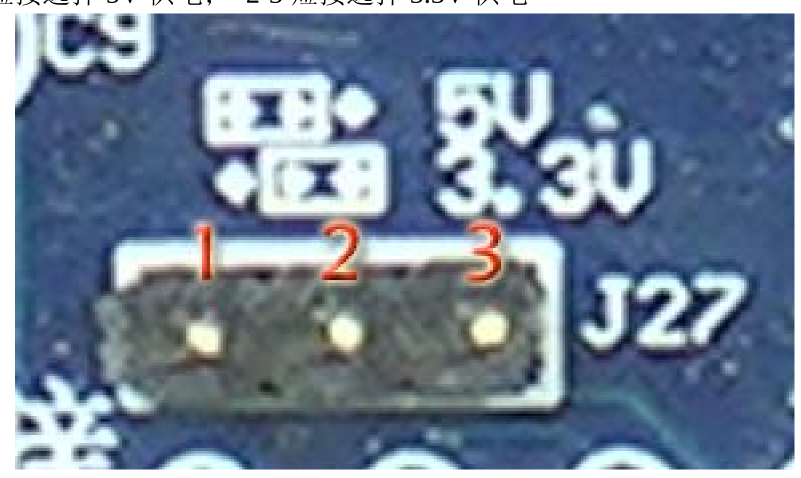


图 3-15 LCD 的电压选择跳线

4.2.12 JTAG 调试接口

SBC-2410X 采用间距 2.0mm 的 20 针 JTAG 接口，如图 3-15 所示。为了方便与标准 2.54 的某些仿真器连接，还需要配备一个 2.0mm 间距转 2.54mm 间距的 JTAG 转接头，如图 3-16 所示。表 3-9 描述了 20 针 JTAG 每个引脚的定义。

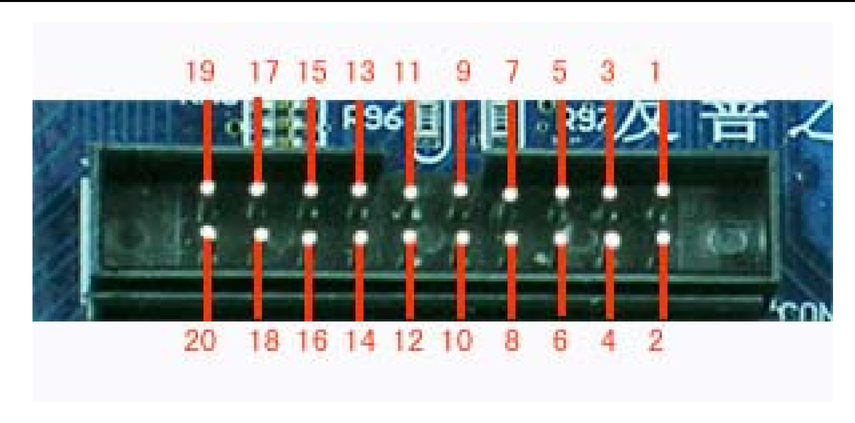


图 3-16 SBC-2410X 之 JTAG 接口(2.0mm 间距)

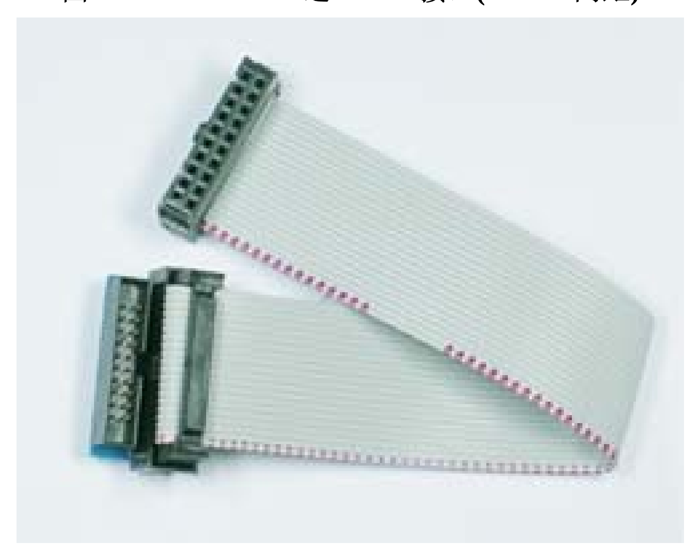


图 3-17　2.0 间距到 2.54 间距 JTAG 转换头

表 3-9 JTAG 引脚定义

序号	引脚定义	序号	引脚定义
1	VDD33V	2	VDD33V
3	nTRST	4	GND
5	TDI	6	GND
7	TMS	8	GND
9	TCK	10	GND
11	下拉电阻	12	GND
13	TDO	14	GND
15	nRESET	16	GND
17	NC	18	GND
19	NC	20	GND

4.2.13 通用 IO 口：CON-GPIO

S3C2410 内部资源众多，SBC-2410X 引出了剩余资源的引脚供用户扩展之用。CON-GPIO 包含 10 路中断，6 路 AD 输入，1 组 SPI 接口等，如下图表所示为。

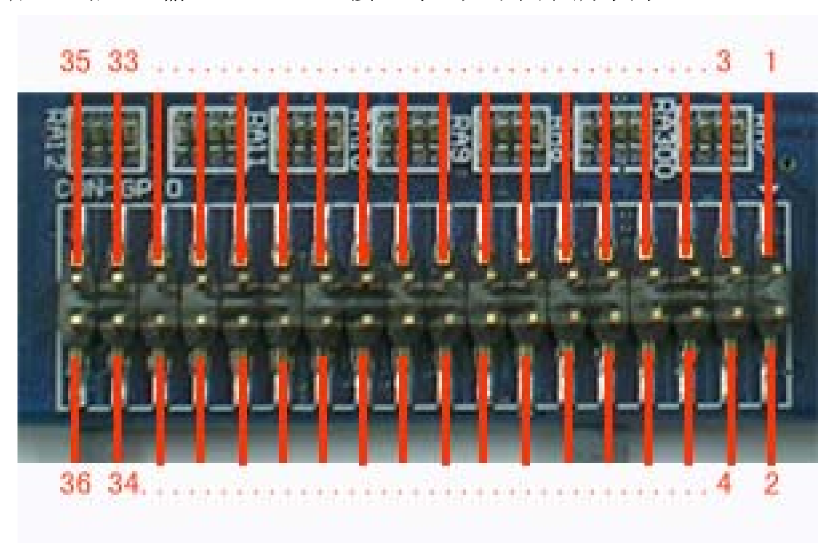


图 3-18　CON-GPIO

表 3-10 CON-GPIO 接口引脚定义

序号	引脚定义	序号	引脚定义
1	VDD33V	2	VDD33V
3	EINT0	4	EINT4
5	EINT5	6	EINT6
7	EINT11	8	EINT13
9	EINT14	10	EINT15
11	EINT16	12	EINT19
13	GND	14	GND
15	I2CSDA	16	I2CSCL
17	CLKOUT0	18	CLKOUT1
19	GPB0	20	GPB1
21	Vref	22	GPB6
23	AIN1	24	AIN0
25	AIN3	26	AIN2
27	AIN6	28	AIN4
29	VDD5V	30	VDD5V
31	SPIMISO	32	SPIMOSI
33	SPICLK	34	nSS_SPI

35	GND	36	GND

4.2.14 音频输入与输出

SBC-2410X 带有一个双声道耳塞音频输出插座和一个单声道音频输入插座。用户可以直接接普通耳塞插座，如下图所示。



图 3-19 音频输入与输出

第五节 Linux 实用软件及工具

4.1 Linux 常用命令

(本小节参考 http://www.tomlinux.com 网站之“**Linux** 开发中的常用命令”一文)

下面是 Linux 常用命令，它们也适合于嵌入式 Linux 系统。

4.1.1 文件列表 – ls

ls #以默认方式显示当前目录文件列表
ls –a #显示所有文件包括隐藏文件
ls –l #显示文件属性，包括大小，日期，符号连接，是否可读写及是否可执行

4.1.2 目录切换 – cd

cd dir #切换到当前目录下的 dir 目录
cd / #切换到根目录
cd .. #切换到到上一级目录

4.1.3 复制 – cp

cp source target #将文件 source 复制为 target
cp /root/source . #将/root 下的文件 source 复制到当前目录
cp –av soure_dir target_dir #将整个目录复制，两目录完全一样
cp –fr source_dir target_dir #将整个目录复制，并且是以非链接方式复制，当 source 目录带有符号链接时，两个目录不相同

4.1.4 删除 – rm

rm file #删除某一个文件

rm -fr dir #删除当前目录下叫 dir 的整个目录

4.1.5 移动 – mv

mv source target #将文件 source 更名为 target

4.1.6 比较 – diff

diff dir1 dir2 #比较目录 1 与目录 2 的文件列表是否相同，但不比较文件的实际内容，不同则列出

diff file1 file2 #比较文件 1 与文件 2 的内容是否相同，如果是文本格式的文件，则将不相同的内容显示，如果是二进制代码则只表示两个文件是不同的

4.1.7 回显 – echo

echo message #显示一串字符

echo "message message2" #显示不连续的字符串

4.1.8 容量查看 – du

du #计算当前目录的容量

du -sm /root #计算/root 目录的容量并以 M 为单位

4.1.9 文件内容查看 – cat

cat file #显示文件的内容，和 DOS 的 type 相同

cat file | more #显示文件的内容并传输到 more 程序实现分页显示，使用命令 less file 可实现相同的功能

4.1.10 分页查看 – more

more #分页命令，一般通过管道将内容传给它，如 ls | more

4.1.11 时间日期 – date

date #显示当前日期时间
date -s 20:30:30 #设置系统时间为 20:30:30
date -s 2002-3-5 #设置系统时期为 2002-3-5

4.1.12 查找 – find

find -name /path file #在/path 目录下查找看是否有文件 file

4.1.13 搜索 – grep

grep -ir “chars” #在当前目录的所有文件查找字串 chars，并忽略大小写，-i 为大小写，-r 为下一级目录

4.1.14 设置环境变量 – export

export LC_ALL=zh_CN.GB2312 #将环境变量 LC_ALL 的值设为 zh_CN.GB2312

4.1.15 编辑 – vi

vi file #编辑文件 file
vi 原基本使用及命令：
输入命令的方式为先按 ctrl+c， 然后输入:x(退出),:x!(退出并保存) :w(写入文件),:w!(不询问方式写入文件)，:r file(读文件 file) ,:%s/oldchars/newchars/g(将所有字串 oldchars 换成 newchars) 这一类的命令进行操作

4.1.16 压缩与解压 – tar

tar xfzv file.tgz #将文件 file.tgz 解压
tar cfzv file.tgz source_path #将文件 source_path 压缩为 file.tgz

4.1.17 挂接 – mount

mount -t yaffs /dev/mtdblock/0 /mnt #把/dev/mtdblock/0 装载到 /mnt 目录

mount -t nfs 192.168.0.1:/friendly-arm/root /mnt #将 nfs 服务的共享目录/friendly-arm/root 挂接到/mnt 目录

4.1.18 启动信息显示 – dmesg

dmesg #显示 kernle 启动及驱动装载信息

4.1.19 改变文件权限 – chmod

chmod a+x file #将 file 文件设置为可执行，脚本类文件一定要这样设置一个，否则得用 bash file 才能执行

chmod 666 file #将文件 file 设置为可读写

4.1.20 创建节点 – mknod

mknod /dev/tty1 c 4 1 #创建字符设备 tty1,主设备号为 4，从设备号为 1，即第一个 tty 终端

4.1.21 进程查看 – ps

ps #显示当前系统进程信息

ps –ef #显示系统所有进程信息

4.1.22 杀死进程 – kill

kill -9 500 #将进程编号为 500 的程序杀死

4.2 SBC 实用测试程序

以下程序均是进入系统后，基于控制台操作。

4.2.1 LED 跑马灯计数器 – led-player

开机进入系统后，将会自动运行运行一个 LED 服务程序(**/etc/rc.d/init.d/leds**)，它其实是调用了 led-player 的一个脚本，led-player 开始运行后，将会在/tmp 目录下创建一个 **led-control** 管道文件，向该管道发送不同的参数可以改变 led 的闪烁模式：

#echo 0 0.2 > /tmp/led-control

运行该命令后，4 个用户 led 将会以每个间隔 0.2 秒的时间运行跑马灯。

#echo 1 0.2 >/tmp/led-control

运行该后，4 个用户 led 将会以间隔 0.2 秒的时间运行累加器。

4.2.2 LED 单独控制 – led

led 是一个可以控制单个 led 的实用程序，要使用 leds 必须先停止 led-player，如下命令：

#/etc/rc.d/init.d/leds stop

该命令将停止 led-player 对 led 的操纵。led 的使用方法如下：

[root@fa /]# **led**

Usage: leds led_no 0|1

led_no 是要操作的 led(可为 0，1，2，3)，0 和 1 分别代表关闭和点亮。

#led 2 1

将点亮 LED3

4.2.3 按键测试 – buttons

buttons 是一个简单的读取 SBC-2410X 上按键的程序。

运行 buttons 后，当按下 K1 时，将打印“1”，松开时打印“129”;

运行 buttons 后，当按下 K2 时，将打印“2”，松开时打印“130”;

运行 buttons 后，当按下 K3 时，将打印“3”，松开时打印“131”;

运行 buttons 后，当按下 K4 时，将打印“4”，松开时打印“132”。

松开后的键值是按下的键值与“0x80”相或后得到的。

4.2.4 控制台下的 Mp3 播放器 – madplay

madplay 是友善之臂移植的一个基于控制台下的 mp3 播放器。它有多种播放控制模式，可以运行“madplay -h”查看其使用帮助，如下：

#madplay -h

```
Usage: madplay [OPTIONS] FILE [...]
Decode and play MPEG audio FILE(s).

Verbosity:
  -v, --verbose              show status while decoding
  -q, --quiet                be quiet but show warnings
  -Q, --very-quiet           be quiet and do not show warnings
      --display-time=MODE    use default verbose time display MODE
                               (remaining, current, overall)

Decoding:
      --downsample           reduce sample rate 2:1
  -i, --ignore-crc           ignore CRC errors
      --ancillary-output=PATH   write ancillary data to PATH

Audio output:
  -o, --output=[TYPE:]PATH   write output to PATH with format TYPE (below)
  -b, --bit-depth=DEPTH      request DEPTH bits per sample
  -R, --sample-rate=HERTZ    request HERTZ samples per second
  -d, --no-dither            do not dither output PCM samples
      --fade-in[=DURATION]   fade-in songs over DURATION (default 0:05)
  -a, --attenuate=DECIBELS   attenuate signal by DECIBELS (-)
  -a, --amplify=DECIBELS     amplify signal by DECIBELS (+)
  -A, --adjust-volume=DECIBELS override per-file volume adjustments

Channel selection:
  -1, --left                 output first (left) channel only
  -2, --right                output second (right) channel only
  -m, --mono                 mix left and right channels for monaural output
  -S, --stereo               force stereo output

Playback:
```

```
  -s, --start=TIME          skip to begin at TIME (HH:MM:SS.DDD)
  -t, --time=DURATION       play only for DURATION (HH:MM:SS.DDD)
  -z, --shuffle             randomize file list
  -r, --repeat[=MAX]        play files MAX times, or indefinitely
      --tty-control         enable keyboard controls
      --no-tty-control      disable keyboard controls

Miscellaneous:
  -V, --version             display version number and exit
      --license             show copyright/license message and exit
  -h, --help                display this help and exit

Supported output formats:
  cdda    CD audio, 16-bit 44100 Hz stereo PCM (*.cdr, *.cda)
  aiff    Audio IFF, [16-bit] PCM (*.aif, *.aiff)
  wave    Microsoft RIFF/WAVE, [16-bit] PCM (*.wav)
  snd     Sun/NeXT audio, 8-bit ISDN mu-law (*.au, *.snd)
  raw     binary [16-bit] host-endian linear PCM
  hex     ASCII hexadecimal [24-bit] linear PCM
  null    no output (decode only)
```

最简单的使用方法是：

#madplay your.mp3

该命令将以缺省模式播放 your.mp3 文件。

4.2.5 使用移动存储设备

SBC-2410X 中，移动存储设备对应的设备文件是**/dev/scsi/host1/bus0/target0/lun0/part***，为了能够和标准 Linux 系统中的优盘设备名兼容，这里创建一个连接

#ln –s /dev/scsi/host1/bus0/target0/lun0/part1 /dev/sda1

该命令已经写入/etc/init.d/rcS 启动脚本中，因此系统启动后就已经可以直接使用/dev/sda1 了。当优盘接入 USB HOST 口(或者接入 USB HUB)，可以用以下命令把优盘设备挂接到系统的某一个目录上：

#mount /dev/sda1 /mnt

表示把优盘挂接到了/mnt 目录。

4.4.6 通过 USB 摄像头抓图 – vidcat

vidcat 是友善之臂移植的一个基于字符界面的摄像头图象抓取程序，运行“vidcat -h”可以查看它的详细使用信息：

```
[root@fa /]# vidcat -h
vidcat: invalid option -- h
VidCat, Version 0.7.2
Usage: vidcat <options>
 -b                            make a raw PPM instead of an ASCII one
 -d <device>                   video device (default: /dev/video)
 -f {ppm|jpeg|png|yuv4mpeg}    output format of the image
 -g                            greayscale instead of color
 -i {tv|comp1|comp2|s-video}   which input channel to use
 -l                            loop on, doesn't make sense in most cases
 -n {pal|ntsc|secam}           select video norm
 -o <file>                     write output to file instead of stdout
 -p c|g|y|Y                    videopalette to use
 -q <quality>                  only for jpeg: quality setting (1-100, default: 80)

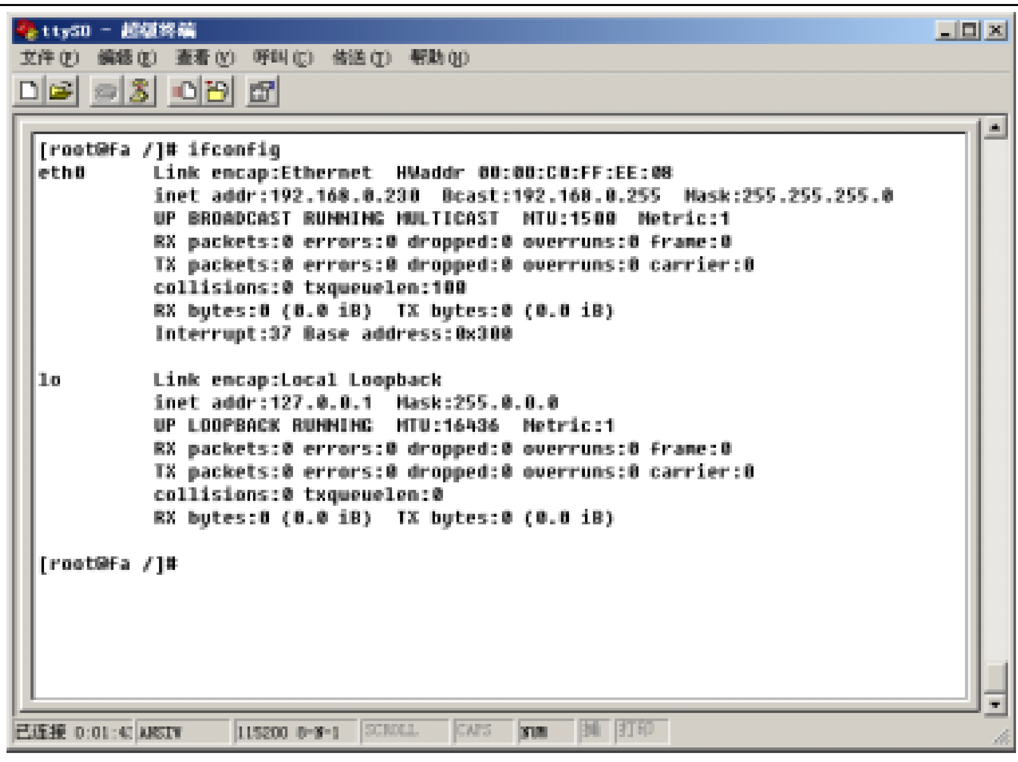
 -s NxN                        define size of the output image (default: 320x240)
Example: vidcat | xsetbg stdin
```

4.3 网络服务及应用程序

本小节介绍如何利用各种命令设置及检测网络环境。

4.3.1 ifconfig 命令

ifconfig 命令可检查并设置主机的网络接口，在 SBC-2410X 上执行时不加任何参数会显示网卡与本地回路(lo,，loopback)的信息，如下图所示：



运行不带参数的 **ifconig**

ifconfig 命令除了可查看网卡的状态外，还能改变一些网络的设置：

#ifconfig eth0 192.168.0.230 netmask 255.255.255.0

表示设置网卡 1 的地址 192.168..230，掩码为 255.255.255.0，不写 netmask 参数则默认为 255.255.255.0。SBC-2410X 缺省网络 IP 地址为 192.168.0.230，这是通过在启动脚本/etc/init.d/rcS 文件中使用 ifconfig 实现的。

也可以利用 ifconfig 命令暂时关闭网卡：

#ifconfig eth0 down

route 是 Linux 系统中控制台下用来设置网关的常用工具，它的使用方法如下：

#route

显示当前路由设置情况

#route add default gw 192.168.0.1

表示设置 192.168.1.1 为默认的路由

#route del default

表示将默认的路由删除

4.3.2 ping 命令

ping 命令可以用来测试本机和网络上的另一台计算机是否连通。

ping –c 3 192.168.0.80

表示向 192.168.0.80 连续发送三次测试包，以验证网络是否连接正常，如果连接正常，则结果如下图所示：

```
[root@fa /]# ping -3 192.168.0.80
BusyBox v0.60.5 (2003.09.05-09:25+0000) multi-call binary

Usage: ping [OPTION]... host

Send ICMP ECHO_REQUEST packets to network hosts.

Options:
        -c COUNT        Send only COUNT pings.
        -s SIZE         Send SIZE data bytes in packets (default=56
        -q              Quiet mode, only displays output at start
                        and when finished.

[root@fa /]# ping -c 3 192.168.0.80
PING 192.168.0.80 (192.168.0.80): 56 data bytes
64 bytes from 192.168.0.80: icmp_seq=0 ttl=128 time=0.9 ms
64 bytes from 192.168.0.80: icmp_seq=1 ttl=128 time=0.4 ms
64 bytes from 192.168.0.80: icmp_seq=2 ttl=128 time=0.4 ms

--- 192.168.0.80 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.9 ms
[root@fa /]#
```

ping –c 192.168.0.80 的结果

4.3.4 traceroute 命令

traceroute 命令可用来检测本机连到其他主机时，数据包在传送过程中经过了哪些路由器及时间。

4.3.5 远程登陆 – telnet

telnet 是一个经常被使用的远程登录工具，关于 telnet 的详细功能和用法可参考网络相关资料。

使用 telnet 登录局域网

当 SBC-2410X 所连接的局域网中，有一台机器提供了 telnet 服务，您就可以使用 SBC-2410X 所带的 telnet 登录该机器了，当然，您必须有相应的帐号和密码。

假设提供了 telnet 服务的机器的 IP 地址为 192.168.0.1，使用以下命令登录该机器：

telnet 192.168.0.1

键入用户名和密码就可以远程使用该机器了。

使用 telnet 登录 bbs

当 SBC-2410X 所连接的网络可以通到互联网，您可以配置 SBC-2410X 登录外面的 bbs，使用 ifconfig 命令配置 SBC-2410X 的 IP 地址，使用 route 命令配置网关：

route add default gw 192.168.0.1

下图是使用 SBC-2410X 登录华南木棉 bbs 的界面。

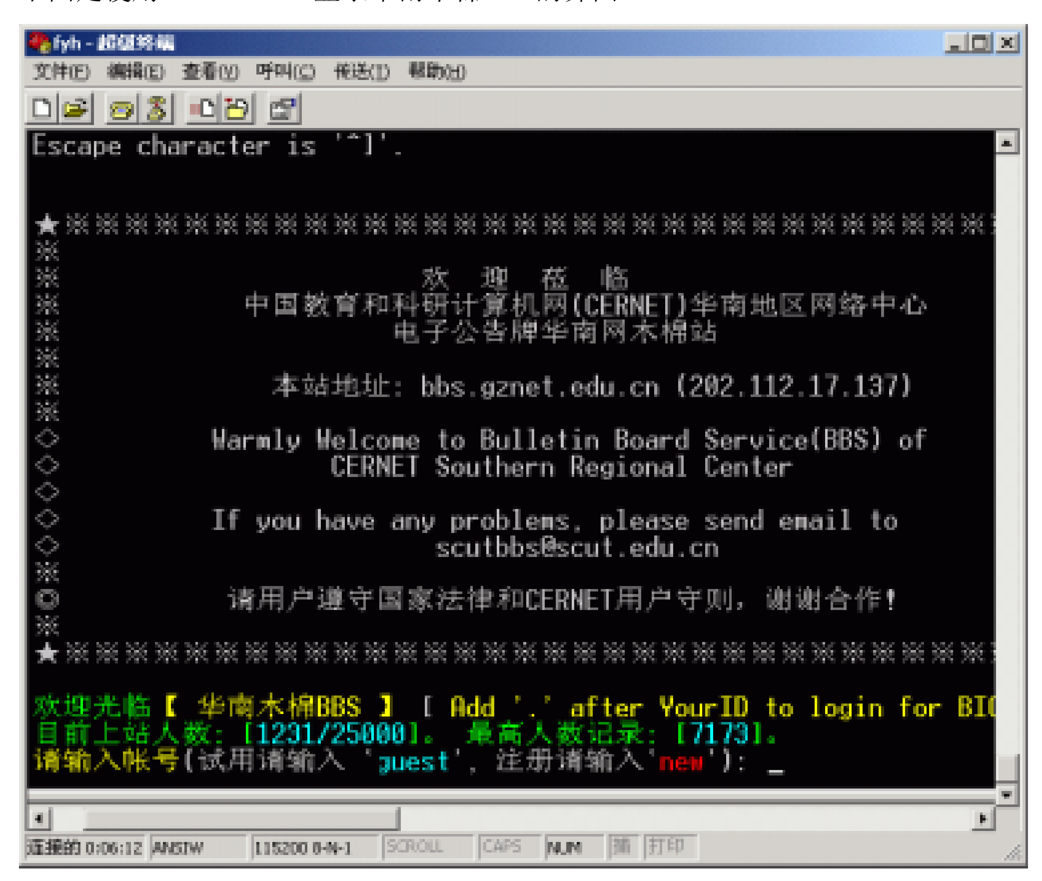


图 4-15 **使用 SBC-2410X 登录华南木棉 bbs**

4.3.6 Telnet 服务器 – inetd

当 SBC-2410X 已经被设置了 IP 地址后，它就可以作为一台 telnet 服务器了，这是因为系统启动的时候已经在后台执行了 inetd 服务程序。

SBC-2410X 的缺省 IP 地址被设置为 192.168.0.230，则在“开始->运行”中敲入以下命令：

telnet 192.168.0.230

输入用户名 **root**，不需要密码按回车，即可登录 SBC-2410X。

4.3.7 远程文件传送 – ftp

ftp 的使用方法与标准 Linux 是相同的，假定您的 ftp 服务器 IP 地址是 192.168.0.80，在 shell 命令提示符下输入：

#ftp 192.168.0.80

如下图所示，就可以正常使用 ftp 的命令了。

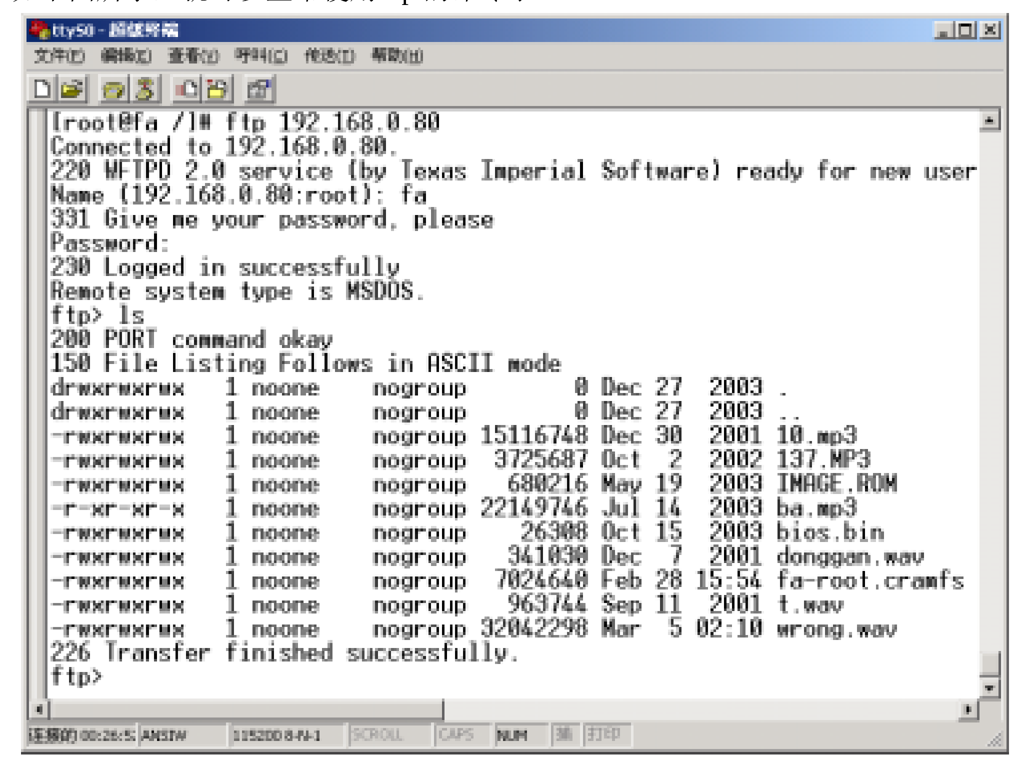


图 4-17 使用 ftp 进行登录

4.3.8 Ftp 服务器 – inetd

当 SBC-2410X 系统正常运行后，将会通过 inetd 启动 ftp 服务，此时 SBC-2410X 可作为一台 ftp 服务器。

在同样网域的计算机上输入：

C:>ftp 192.168..230

用户名：fa

密码：fa

即可登陆 SBC-2410X

4.3.9 Web 服务器 – boa

系统开机后，并且 LED 计数器显示跳动正常，就可以使用 IE 或者其他浏览器来浏览存放于 SBC-2410X 中的网页了，SBC-2410X 的默认 IP 地址是 192.168.0.230，并且自动运行了 web 服务程序(boa)则在浏览器的地址栏中输入：

http://192.168.0.230

此时会跳出如下图所示页面，这表明您已经通过浏览器连接到 SBC-2410X 了。



4.3.10 通过 Web 服务器控制 LED

系统开机后，并且 LED 计数器显示跳动正常，就可以使用 IE 或者其他浏览器来浏览存放于 SBC-2410X 中的网页了，SBC-2410X 的默认 IP 地址是 192.168.0.230，并且自动运行了 web 服务程序(boa)则在浏览器的地址栏中输入：

http://192.168.0.230

此时会跳出如图 4-18 所示页面，您可以使用网页中的各个测试项目进行测试，其中的“LED 测试”将会通过 CGI 程序来控制板上的 LED 灯，其中包括 2 种方式的显示类型和三种不同的显示速度。如图 4-19 所示

图 4-18 使用 IE 浏览器显示 SBC-2410X 中的网页

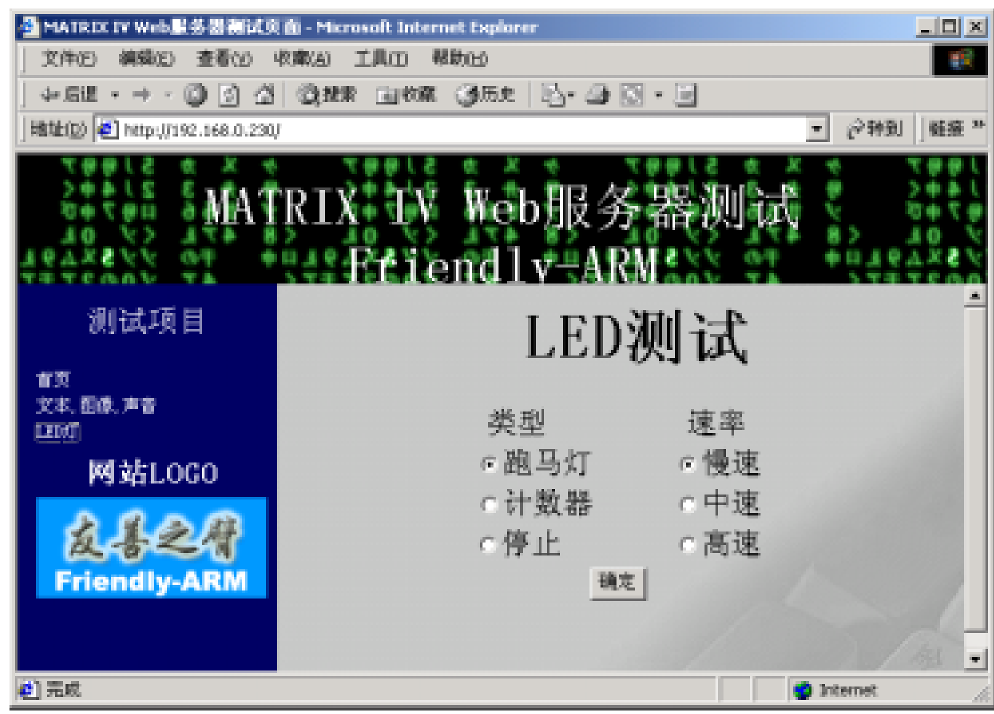


图 4-19 LED 测试内容

如果要停止 web 服务器，则在 shell 命令提示符下输入以下命令，如图 4-20 所示：

#/etc/rc.d/init.d/httpd stop

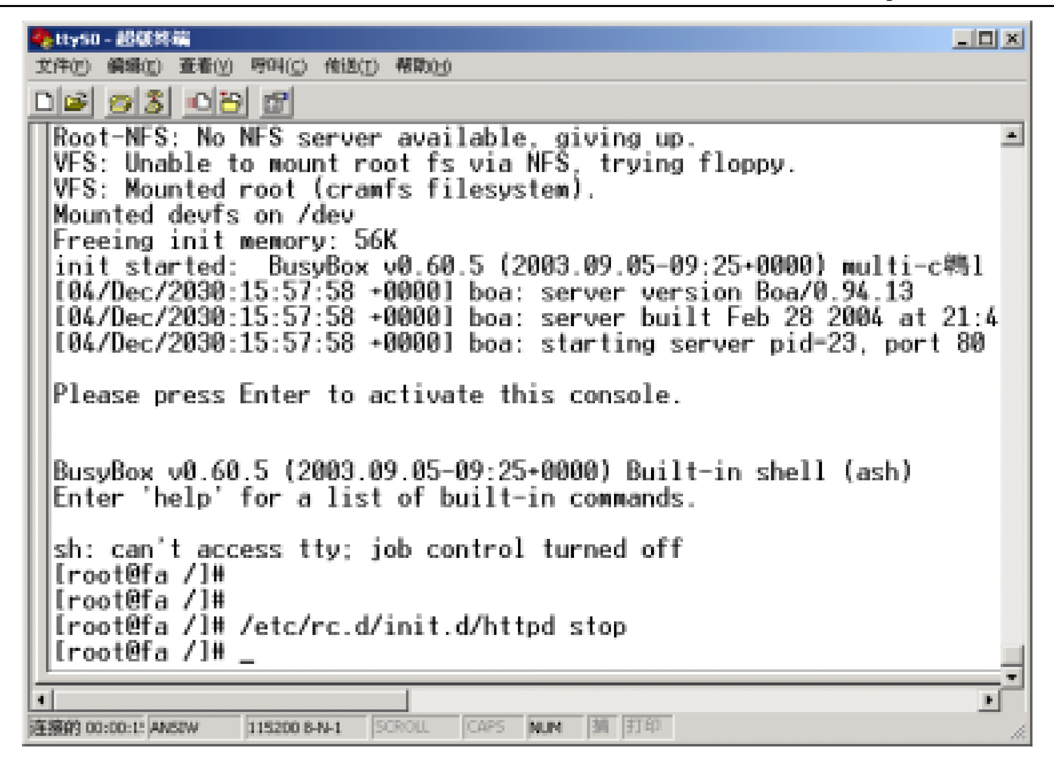


图 4-20 停止 **web** 服务器

要重新启动则输入：

#/etc/rc.d/init.d/httpd start

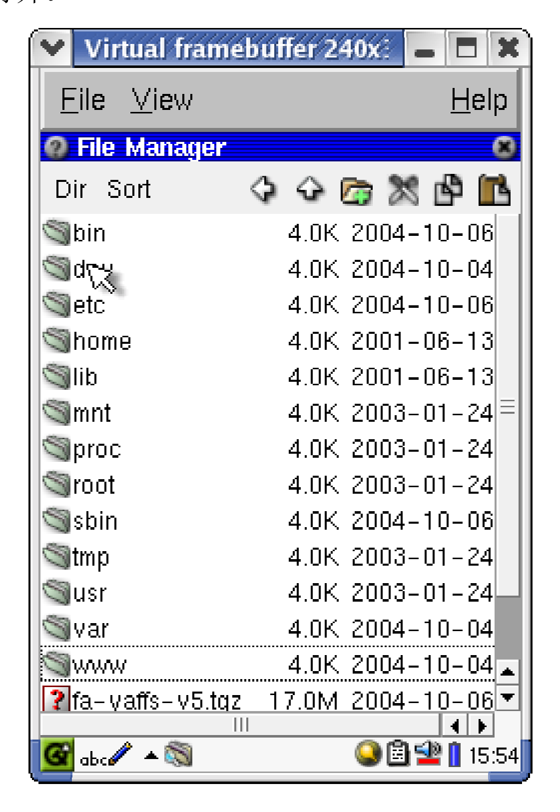
4.4 基于 Qtopia 的图形程序

Qt/Embedded 是一款十分优秀的嵌入式图形界面平台，Qtopia 是基于 Qt 编写的一个用于手持设备的用户信息管理软件，它集成了很多实用的程序。友善之臂基于 Qtopia 开发了几个简单的应用示例，鉴于 Qtopia 在各平台的运行效果是一样的，下面通过运行于 X86 上的屏幕截图来一一认识一下这些程序。

4.4.1 应用程序

文件管理器

文件管理器看起来类似于 Windows 系统中的资源管理器，通过文件和应用程序的关联关系，可以以相应的程序打开。



媒体播放器

Qtopia 附带的媒体播放器可以播放多种文件格式的媒体文件，如 mp3, wav, mpeg，avi 等。下图的中的播放器正在播放/root/Documents 中的一首 mp3，该文件是 SBC-2410X 缺省附带的。



摄像头动态播放

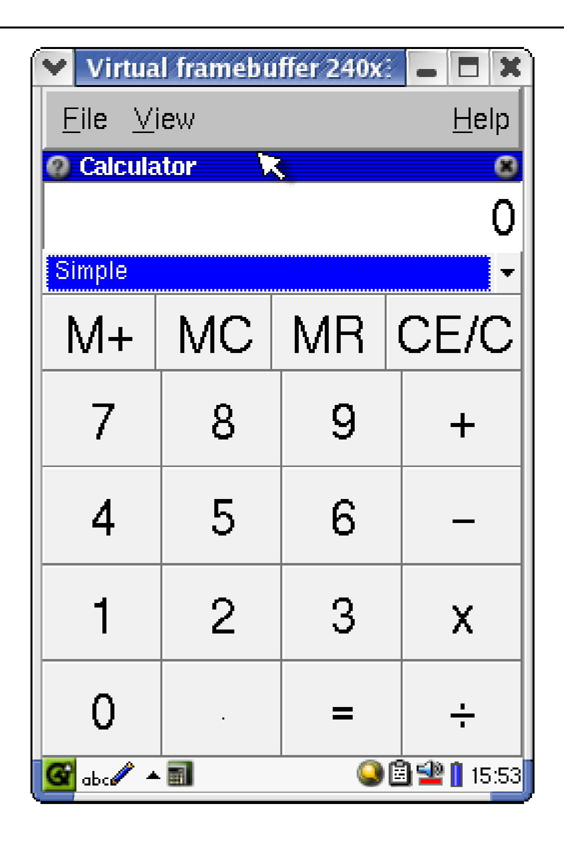
SBC-2410X 带有一个摄像头动态播放软件，名为 **Camra Tracer**，意为“跟踪者”，该软件由友善之臂自主开发，并不是移植的 Open Source 的软件，另外该软件并不开放源代码。



通过 USB 摄像头动态监控

计算器

Qtopia 附带的计算器有简单型和科学型两种模式，下图是简单模式。



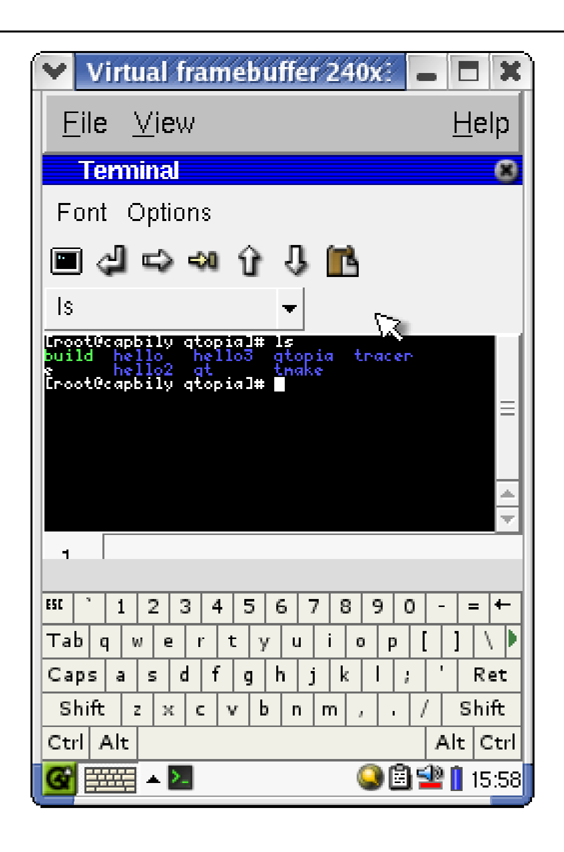
图片浏览器

Qtopia 带的图片浏览器具有翻转、全屏显示，自动浏览等效果，支持 bmp、png、xpm 等格式的图片(可以通过编译选项配置支持的图片格式)。



控制台终端

通过控制台终端可以使用字符模式的各种命令，用户可以通过连接一个 USB 键盘十分方便的输入命令，当没有键盘时，也可以使用 Qtopia 提供的软键盘，下图就是使用软键盘的演示。


Virtual framebuffer 240x
File View Help
Terminal
Font Options
ls
[root@capbily qtopia]# ls
build hello hello3 qtopia tracer
hello2 qt tmake
[root@capbily qtopia]#
15:58

文本编辑器



地址本

Qtopia 提供了一个手持设备中常见的地址薄，用户可以使用它记录联系方式相关的信息。



VNC 客户浏览器

当某台主机配置并启动了 VNC 服务器以后，可以通过 VNC 来连接到该主机，下图是使用 SBC-2410X 带的 VNC 客户端浏览器，运行远程 PC 的 Redhat9.0。



在 SBC-2410X 上使用 VNC 远程运行 Redhat

4.4.2 游戏

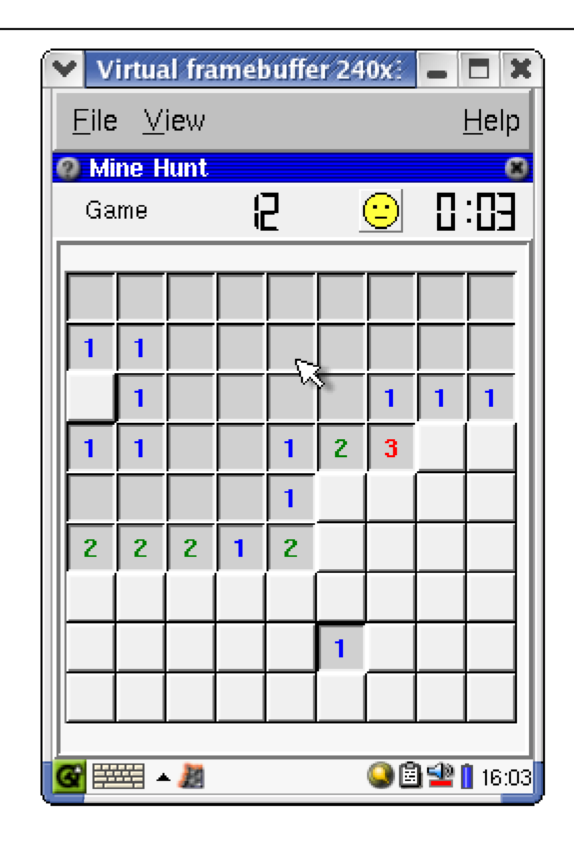
空中大战

这是一个十分有趣的空中击战游戏，三维的界面显示。为了更加方便地控制，建议接一个 USB 键盘，通过 F9 键可以重新开始新一轮游戏，向上地方向键表示启动和前进，空格是发射。



扫雷

扫雷是一个很十分常见的游戏，这里的操作方法可能有点不同。

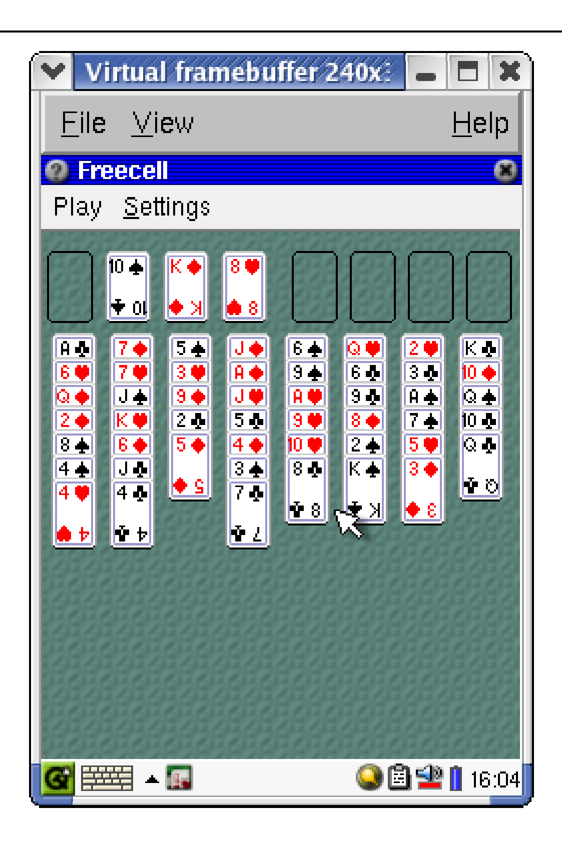


空降兵

该游戏同样需要键盘来配合控制。

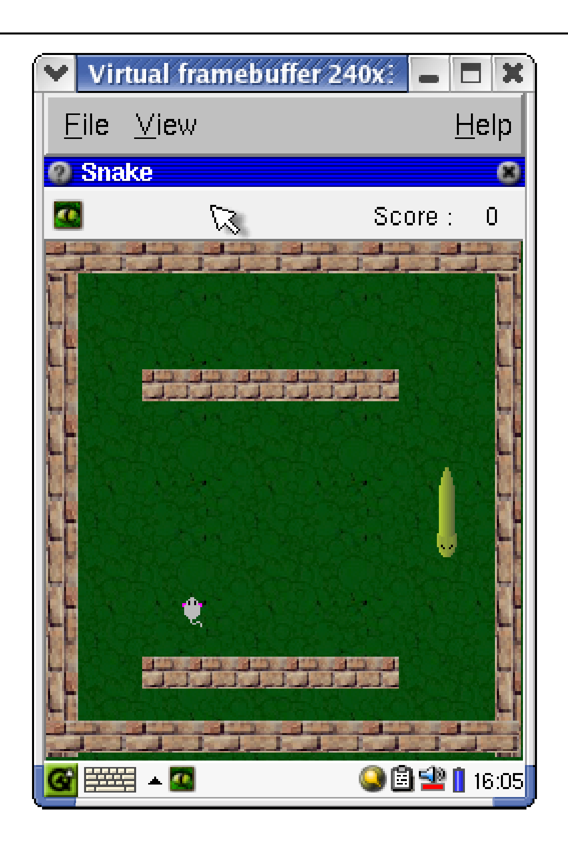
扑克牌

Qtopia 的扑克牌消遣游戏包含 2 种玩法，可以查看相应的帮助了解使用方法。



贪吃蛇

一个常见的手持设备中的游戏，需要有键盘配合控制。



十五点

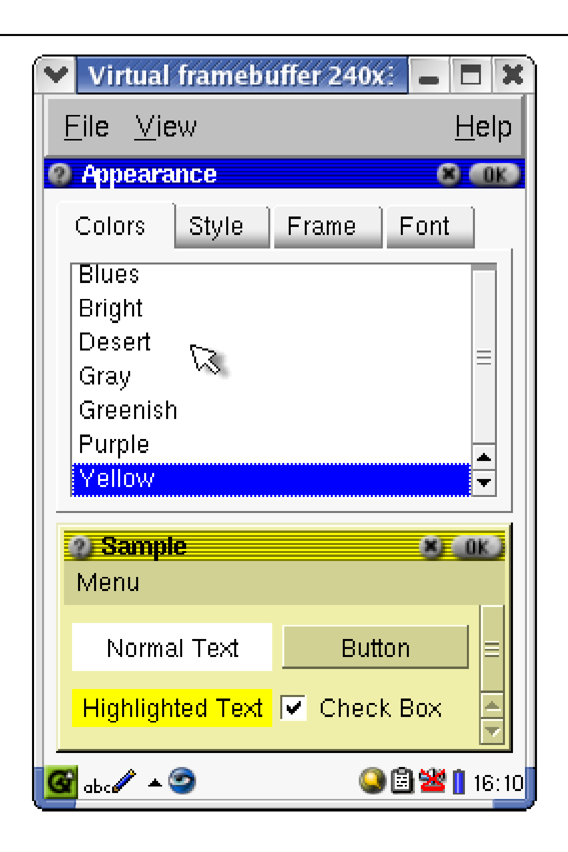
一个常见的方块移动游戏，不需要键盘就可以控制。



4.2.3 系统设置

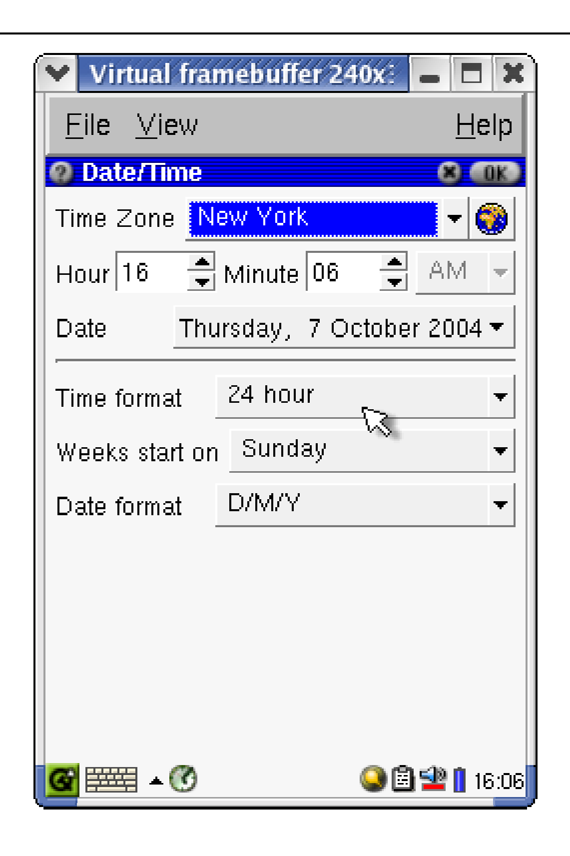
外观设置

通过外观设置可以设置 Qtopia 的主题色彩，字体，窗口风格等。



日期时间

这是用来设置时间的一个程序。



语言设置

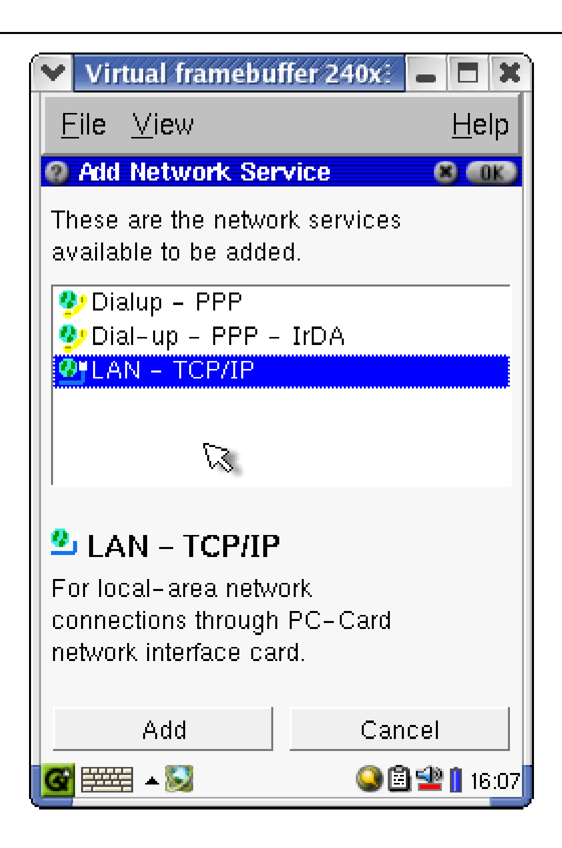
虽然 Qtopia 带了语言设置系统选项，但是，缺省的 Qtopia 对的支持似乎不是太完美。



任务栏设置

网络设置

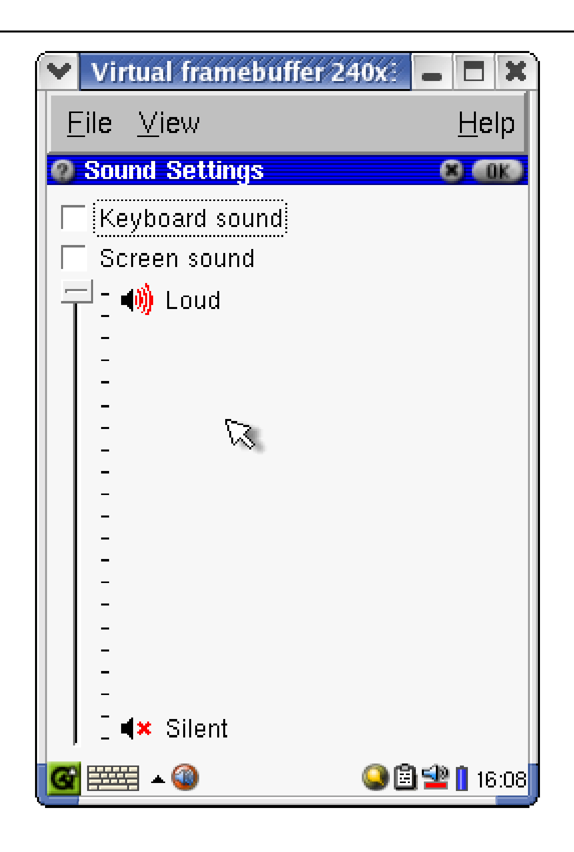
通过“网络设置”可以设置系统的 IP 地址，网关等参数，拨号设置等。



屏幕翻转设置

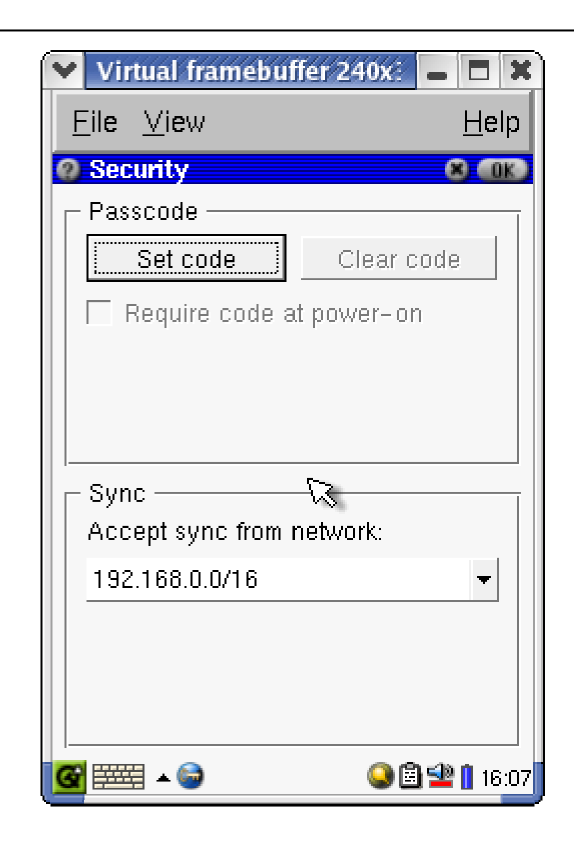
声音设置

如图所示为系统的声音设置



密码设置

通过“密码设置”可以设置 Qtopia 的启动密码。因为 SBC-2410X 使用了可读写的文件系统，所以可以实现此项设置。



4.5 基于 Microwindows 的图形程序

由于存储空间的限制，目前，SBC-2410X 仅带了一个 microwindows 的扫雷程序(mine)作为演示。

因为系统开机默认运行的是 Qtopia，为了能够运行 mine，需要先杀死 Qtopia 进程，然后通过控制终端启动 mine。

#kill qtopia #杀死 qtopia 进程

#mine #运行 mine 扫雷程序



SBC-2410X 运行 Microwindows 程序

第六节 基于 S3C-2410X 的嵌入式 Linux 开发

SBC-2410X(预装嵌入式 Linux)整个根目录使用了可读写的 **yaffs** 文件系统，因此使用 SBC-2410X 开发应用程序十分方便，主要有下面三个步骤：

1) 在主机上建立/安装开发环境(包括编译器，各种源代码及其他工具等)
2) 在主机平台上开发应用程序/驱动程序
3) 在 **SBC-2410X** 上安装应用程序/驱动程序。

6.1 建立开发环境

以下安装步骤基于 **Redhat9.0**。安装开发工具之前请确保您的硬盘有 **1GB** 的剩余空间。
(注：图形界面编程的 SDK 安装见第七节)
以 root 用户登录，把 SBC-2410X 光盘放入光驱中。
Step1：挂接光盘
#mount /dev/cdrom /mnt/cdrom
Step2：创建 friendly-arm 目录
#mkdir /friendly-arm
Step3：进入光盘目录
#cd /mnt/cdrom/SBC-2410-Linux
Step4：安装编译器
#tar xvzf arm-linux-toolchains.tgz –C /
Step5：解压 SBC-2410X 内核源代码
#tar xvzf matrix5-kernel.tgz –C /friendly-arm
Step6：安装 Jtag 烧写程序(含源代码)
#tar xvzf Jflash.tgz –C /friendly-arm
Step7：安装用于作为网络文件系统(nfs)的 root 目录
#tar xzvf root-for-nfs-v5.tgz –C /friendly-arm
Step8：安装 bootloader 源代码
#tar xzvf boot.tgz –C /friendly-arm
Step9：安装应用程序源代码
#tar xzvf example.tgz –C /friendly-arm
Step10：设定网络文件系统共享目录

#cat exports >> /etc/exports

Step11：启动网络文件系统服务

＃**/etc/init.d/nfs restart**

Step12：弹出光盘

#umount /dev/cdrom

#eject

另外，为了可以方便的使用 arm-linux-gcc 编译器系统，建议把 arm-linux 工具链目录加入到环境变量 PATH 中，如图 6-1 所示修改/etc/profile 文件：

```
root@capbily:/friendly-arm/root
File  Edit  View  Terminal  Go  Help
}

# Path manipulation
if [ `id -u` = 0 ]; then
        pathmunge /sbin
        pathmunge /usr/sbin
        pathmunge /usr/local/sbin
        pathmunge /usr/local/arm/2.95.3/bin
fi
添加该句即可把arm-linux-gcc路径加入PATH

pathmunge /usr/X11R6/bin after

unset pathmunge

                                  21,1-8        36%
```

图 6-1 **把 arm-linux** 工具链路径加入环境变量 **PATH**

6.2 应用程序编程指南

6.2.1 Hello, SBC-2410X!

Step1：编辑源代码

在 PC 上编辑以下源代码，并保存为 hello.c

```
#include <stdio.h>

int main(void)
{
 printf("Hello, SBC-2410X\n");
        return 0;
   }
```

Step2：编译 **hello**

使用以下命令编译：

＃**arm-linux-gcc –o hello hello.c**

将生成 **hello** 可执行文件。

Step3：下载并运行

将可执行文件移动到 SBC-2410X 目前主要有两种方式：

(1) 复制到介质(下面以优盘为例)

把优盘插入 PC 的 USB，然后执行以下命令把 hello 复制到优盘

#mount /dev/sda1 /mnt

#cp hello /mnt

#umount /mnt

把优盘拔下来插入到 SBC-2410X 的 USB HOST 端口，按照以下命令操作：

#mount /dev/sda1 /mnt ；挂接优盘

#cp /mnt/hello /bin ；把 hello 复制到 bin 目录

#hello ；执行 hello

(2)通过网络移动(推荐使用)

通过网络下载程序的主要步骤是：先把 hello 复制到 ftp 共享目录，然后在 SBC-2410X 上使用 ftp 下载，并修改执行权限运行，如下：

在 PC 端执行：

#cp hello /home/ftp ；把 hello 复制到 ftp 共享目录

在 SBC-2410X 端执行：

#cd /bin ；进入 bin 目录

#ftp 192.168.0.1 ；登录 ftp 服务器
>**get hello** ；下载 hello
>**bye** ；退出 ftp 登录
#chmod a+x hello ；改变 hello 的可执行权限
#hello ；执行 hello

6.2.2 测试 LED

led 控制程序的源代码位于/friendly-arm/examples/led 目录下，下面是程序清单及注释。

程序清单：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

int main(int argc, char **argv)
{
 int on;
 int led_no;
 int fd;

/* 检查 led 控制的两个参数，如果没有参数输入则退出。*/
 if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d", &on) != 1 ||
        on < 0 || on > 1 || led_no < 0 || led_no > 3) {
        fprintf(stderr, "Usage: leds led_no 0|1\n");
        exit(1);
 }

/*打开/dev/leds 设备文件*/
 fd = open("/dev/leds", 0);
 if (fd < 0) {
        perror("open device leds");
        exit(1);
 }

/*通过系统调用 ioctl 和输入的参数控制 led*/
 ioctl(fd, on, led_no);
```

```
/*关闭设备句柄*/
 close(fd);

return 0;
}
```

你可以按照上面的 hello 程序的步骤手工编译出 led 可执行文件，然后下载到 SBC—2410X 运行它。

6.2.3 测试按键

按键测试程序 buttons 的源代码位于/friendly-arm/examples/buttons 目录下，下面是程序清单及注释：

```
/*
 *        Buttons Example for Matrix V
 *
 *        Copyright (C) 2004 capbily - friendly-arm
 *    capbily@hotmail.com
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
{
 int buttons_fd;
 int key_value;

/*打开键盘设备文件*/
```

```
    buttons_fd = open("/dev/buttons", 0);
    if (buttons_fd < 0) {
        perror("open device buttons");
        exit(1);
    }

    for (;;) {
        fd_set rds;
        int ret;

        FD_ZERO(&rds);
        FD_SET(buttons_fd, &rds);

        /*使用系统调用 select 检查是否能够从/dev/buttons 设备读取数据*/
        ret = select(buttons_fd + 1, &rds, NULL, NULL, NULL);

        /*读取出错则退出程序*/
        if (ret < 0) {
            perror("select");
            exit(1);
        }

        if (ret == 0) {
            printf("Timeout.\n");
        }
        /*能够读取到数据*/
        else if (FD_ISSET(buttons_fd, &rds)) {
            /*开始读取键盘驱动发出的数据，注意 key_value 和键盘驱动中定义为一致的类型
*/
            int ret = read(buttons_fd, &key_value, sizeof key_value);
            if (ret != sizeof key_value) {
                if (errno != EAGAIN)
                    perror("read buttons\n");
                continue;
            } else {
                /*打印键值*/
                printf("buttons_value: %d\n", key_value);
            }
```

```
        }
    }

    /*关闭设备文件句柄*/
    close(buttons_fd);
    return 0;
}
```

你可以按照上面的 hello 程序的步骤手工编译出 buttons 可执行文件，然后下载到 SBC—2410X 运行它

6.2.4 UDP 网络编程

TCP/IP 提供了无连接的传输层协议：UDP(User Datagram Protocol，即用户数据报协议)。UDP 与 TCP 有很大的区别，因为无连接的 socket 编程与面向连接的 socket 编程也有很大的差异。由于不用建立连接，因此每个发送个接收的数据报都包含了发送方和接收方的地址信息。

在发送和接收数据之前，先要建立一个数据报方式的套接字，该 socket 的类型为 SOCK_DGRAM，用如下的调用产生：

sockfd=socket(AF_INET, SOCK_DGRAM, 0);

由于不需要建立连接，因此产生 socket 后就可以直接发送和接收了。当然，要接收数据报也必须绑定一个端口，否则发送方无法得知要发送到哪个端口。Sendto 和 recvfrom 两个系统调用分别用于发送和接收数据报，其调用格式为：

int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

int recvfrom(int, s, void *buf, int len, unsigned int flags, struct sockaddr *from, int fromlen);

其中 s 为所使用的 socket，msg 和 buf 分别为发送和接收的缓冲区指针，len 为缓冲区的长度，flags 为选项标志，此处还用不到，设为 0 即可。to 和 from 就是发送的目的地址和接收的来源地址，包含了 IP 地址和端口信息。tolen 和 fromlen 分别是 to 和 from 这两个 socket 地址结构的长度。这两个函数的返回值就是实际发送和接收的字节数，返回-1 表示出错。

使用无连接方式通信的基本过程如图 6-2 所示。

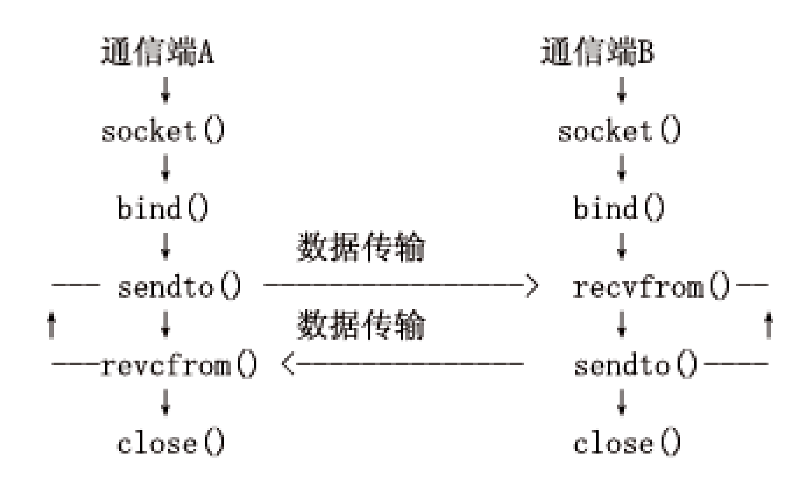


图 6-2 UDP 通信的基本过程

上图描述的是通信双方都绑定自己地址端口的情形，但在某些情况下，也可能有一方不用绑定地址和端口。不绑定的一方的地址和端口由内核分配。由于对方无法预先知道不绑定的一方的端口和 IP 地址(假设主机有多个端口，这些端口分配了不同的 IP 地址)，因此只能由不绑定的一方先发出数据报，对方根据收到的数据报中的来源地址就可以确定回送数据报所需要的发送地址了。显然，在这种情况下对方必须绑定地址和端口，并且通信只能由非绑定方发起。

与 read()和 write()相似，进程阻塞在 recvfrom()和 sendto()中也会发生。但与 TCP 方式不同的是，接收到一个字节数为 0 的数据报是有可能的，应用程序完全可以将 sendto()中的 msg 设为 NULL，同时将 len 设为 0。

下面是一个基于以上原理分析的一个 udp 编程的例子(源代码位于/friendly-arm/examples/udptalk 目录)：

```
/*
 *        udptalk  :  Example for Matrix V
 *
 *        Copyright (C) 2004 capbily - friendly-arm
 *    capbily@hotmail.com
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>
```

```
#define BUFLEN 255

int main(int argc, char **argv)
{
 struct sockaddr_in peeraddr, /*存放谈话对方 IP 和端口的 socket 地址*/
                       localaddr;/*本端 socket 地址*/
 int sockfd;
 char recmsg[BUFLEN+1];
 int socklen, n;

 if(argc!=5){
        printf("%s <dest IP address> <dest port> <source IP address> <source port>\n", argv[0]);
        exit(0);
 }

 sockfd = socket(AF_INET, SOCK_DGRAM, 0);
 if(sockfd<0){
        printf("socket creating err in udptalk\n");
        exit(1);
 }
 socklen = sizeof(struct sockaddr_in);
 memset(&peeraddr, 0, socklen);
 peeraddr.sin_family=AF_INET;
 peeraddr.sin_port=htons(atoi(argv[2]));
 if(inet_pton(AF_INET, argv[1], &peeraddr.sin_addr)<=0){
        printf("Wrong dest IP address!\n");
        exit(0);
 }
 memset(&localaddr, 0, socklen);
 localaddr.sin_family=AF_INET;
 if(inet_pton(AF_INET, argv[3], &localaddr.sin_addr)<=0){
        printf("Wrong source IP address!\n");
        exit(0);
 }
 localaddr.sin_port=htons(atoi(argv[4]));
 if(bind(sockfd, &localaddr, socklen)<0){
        printf("bind local address err in udptalk!\n");
```

```
        exit(2);
    }

    if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
    if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
        printf("sendto err in udptalk!\n");
        exit(3);
    }

    for(;;){
        /*recv&send message loop*/
        n = recvfrom(sockfd, recmsg, BUFLEN, 0, &peeraddr, &socklen);
        if(n<0){
            printf("recvfrom err in udptalk!\n");
            exit(4);
        }else{
            /*成功接收到数据报*/
            recmsg[n]=0;
            printf("peer:%s", recmsg);
        }
        if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
        if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
            printf("sendto err in udptalk!\n");
            exit(3);
        }
    }
}
```

将 udptalk.c 编译好后就可以运行了，/friendly-arm/exampls/udptalk 目录下的 Makefile 指定了两个编译目标可执行文件，一个用于在主机端的 x86-udptalk，一个是用于 SBC-2410X 的 arm-udptalk，运行 make 命令将把这两个程序一起编译出来。可以把 arm-udptalk 使用上面介绍的方法下载到 SBC-2410X 中(预装的 Linux 不含该程序)，假设主机的 IP 地址为 192.168.0.1，SBC-2410X 的 IP 地址为 192.168.0.230。

在主机的终端上输入：

#./x86-udptalk 192.168.0.230 2000 192.168.0.1 2000

在 SBC-2410X 上的终端输入

#arm-udptalk 192.168.0.1 2000 192.168.0.230 2000

则运行结果分别如图 6-3、图 6-4 所示：

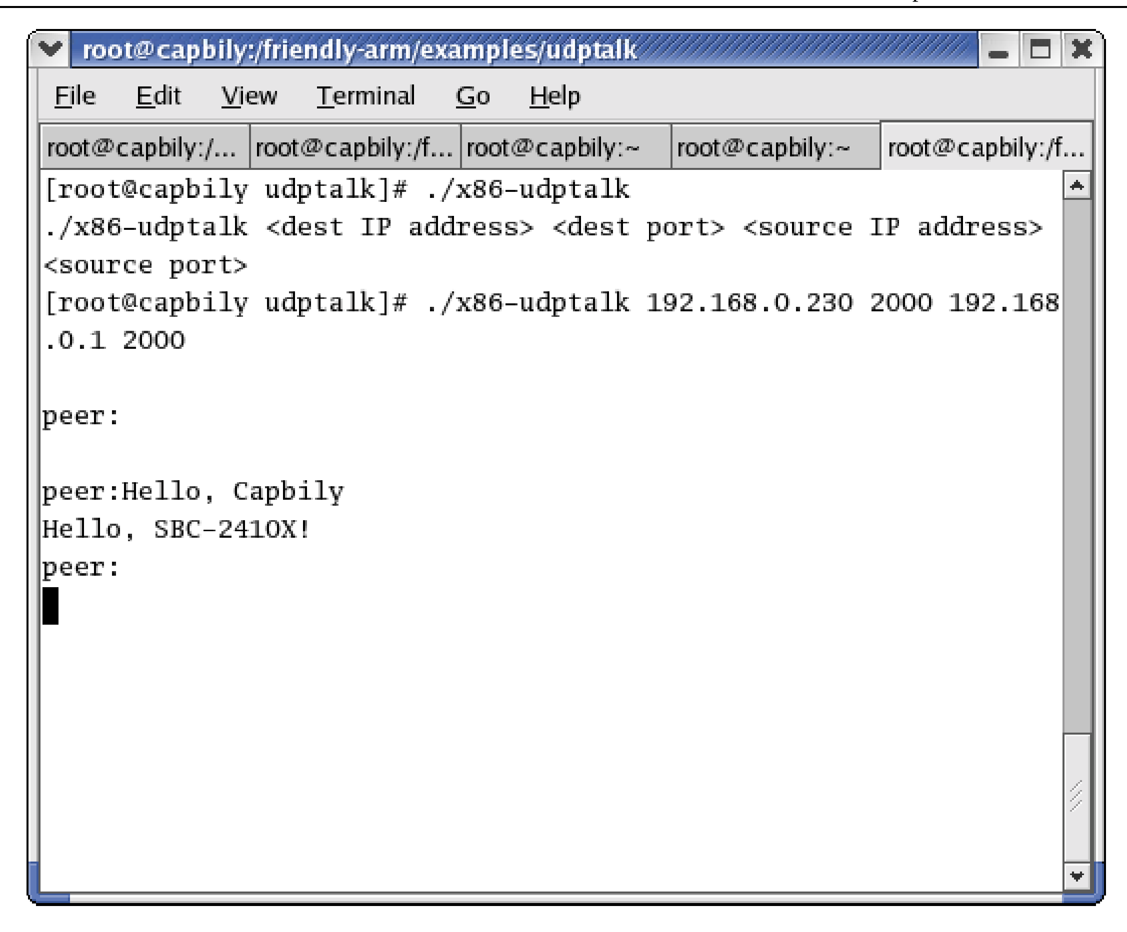


图 **6-3** 在主机上运行 **x86-udptalk**

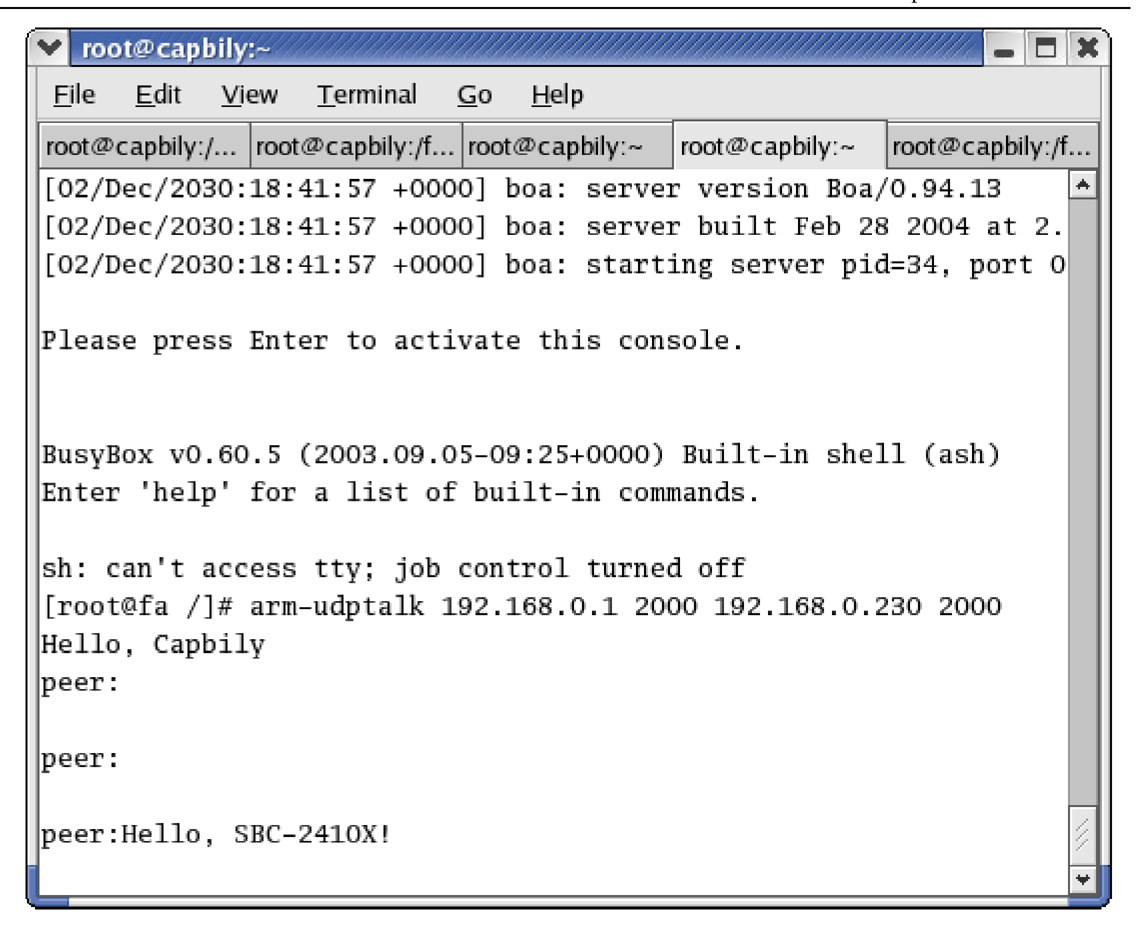


图 **6-4** 在 **SBC-2410X** 上运行 **arm-udptalk**

6.2.5 音频应用程序

(待续)

6.2.6 USB 摄像头抓图程序

(待续)

6.3 Linux 驱动程序开发指南

Linux 操作系统将所有的设备(而不仅是存储器里的文件)全部都看成文件，都纳入文件系统的范畴，都通过文件的操作界面进行操作。这意味着：

- 每一个设备都至少由文件系统的一个文件代表，因而都有一个“文件名”。每个这样的“设备文件”都唯一地确定了系统中地一项设备。应用程序通过设备地文件寻找访问具体地设备，而设备则象普通文件一样受到文件系统访问权限控制机制地保护。
- 应用程序通常可以通过系统调用 open()“打开”这个设备文件，建立起与目标设备的连接。代表着该设备的文件节点中记载着建立这种连接所需的信息。对于执行该应用程序的进程而言，建立起的连接就表现为一个已经打开的文件。
- 打开了代表着目标设备的文件，即建立起与设备的连接后，就可以通过 read()、write()、ioctl()等常规的文件操作对目标设备进行操作。

Linux 将设备分成两大类。一类是像磁盘那样以记录块或“扇区”为单位，成块进行输入/输出设备，称为“块设备”；另一类是像键盘那样以字符(字节)为单位，逐个进行输入/输出的设备，称为“字符设备”、文件系统通常都建立在块设备上。网路设备是介于块设备和字符设备之间的一种特殊设备。

设备文件的属性由三部分信息组成：第一部分是文件的类型(c/b)，第二部分是一个“主设备号”，第三部分是一个“次设备号”。其中设备类型和主设备号结合在一起唯一地确定了设备文件地驱动程序及其界面，而次设备号则说明目标设备是同类设备中的第几个。

6.3.1 Linux 设备驱动程序概述

在 Linux 操作系统中，驱动程序是操作系统内核与硬件设备的直接接口，驱动程序屏蔽了硬件的细节，驱动程序是内核的一部分，他完成以下功能：

- 对设备初始化和释放
- 对设备进行管理，包括实时参数设置以及提供对设备的操作接口
- 读取应用程序传送给设备文件的数据并回送应用程序请求的数据
- 检测是处理设备出现的错误。

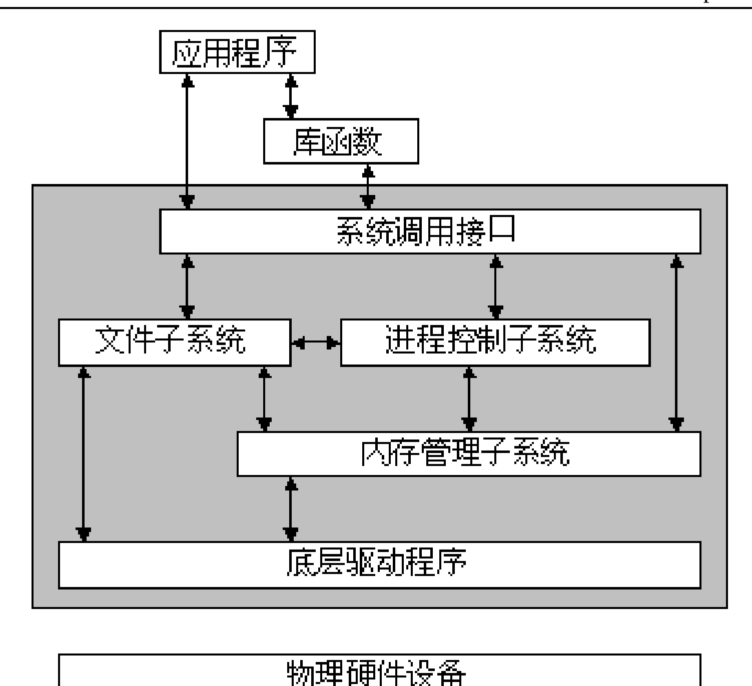


图 6-5 Linux 内核体系结构

如图 6-5 所示，应用程序通过 Linux 的系统调用与内核通信。由于 Linux 中将设备当作文件处理，所以对设备进行操作的调用和对文件操作的操作类似，主要包括 open()、read()、write()、ioctl()、close()等。应用程序发出系统调用命令后，会从用户态转到内核态，通过内核将 open()这样的系统调用转换成对物理设备的操作。

在 Linux 中通过分层实现对物理设备的调用，并使得内核的结构清晰，提高了模块化的独立性。

6.3.1.1 驱动程序的结构

一般 Linux 设备驱动程序可以分为 3 个主要组成部分：

(1)自动配置和初始化子程序，负责检测所要驱动的硬件设备是否存在和能否正常工作。如果设备正常则对这个设备及其相关的设备驱动程序需要的软件状态进行初始化。这部分驱动程序仅在初始化时被调用一次。

(2)服务于 I/O 请求的子程序，又称为驱动程序的上半部。调用这部分程序是由于系统调用的结果。这部分程序在执行时，系统仍认为是与进行调用的进程属于同一个进程，只是由用户态变成了核心态，具有进行此系统调用的用户程序的运行环境，因而可以在其中调用 sleep()等与进程运行环境有关的函数。

（3）中断服务程序，又称为驱动程序的下半部。在 Linux 系统中并不是直接从中断向量表调用设备驱动程序的中断服务子程序，而是由 Linux 系统来接收硬件中断，再由系统调用中断服务子程序。中断可以在任何一个进程运行时产生，因而在中断服务程序被调用时，不能依赖于任何进程的状态，也就不能调用任何与进程运行环境有关的函数。因为设备驱动程序一般支持同一类型的若干设备，所以一般在系统调用中断服务子程序时，都带有一个或多个参数，

以唯一标志请求服务的设备。

在系统内部，I/O 设备的存/取通过一组固定的入口点来进行，这组入口点是由每个设备的设备驱动程序提供的。具体到 Linux 系统，设备驱动程序所提供的这组入口点由一个文件操作结构来向系统进行说明。file_operations 结构定义于 linux/fs.h 文件中，随着内核的不断升级，file_operations 结构也越来越大，不同版本的内核会稍有不同。

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long, unsigned
long, unsigned long);
};
```

file_operations 结构中的成员全部是函数指针，所以实质上就是函数跳转表。每个进程对设备的操作，都会根据 major、minor 设备号，转换成对 file_operations 结构的访问。

常用的操作包括以下几种：

lseek，移动文件指针的位置，只能用于可以随机存取的设备。

read，进行读操作，参数 buf 为存放读取结果的缓冲区，count 为所要读取的数据长度。返回值为负表示读取操作发生错误；否则，返回实际读取的字节数。对于字符型，要求读取的字节数和返回的实际读取字节数都必须是 inode-I_blksize 的倍数。

write，进行写操作，与 read 类似。

select，进行选择操作。如果驱动程序没有提供 select 入口，select 操作将会认为已经准备好进行任何的 I/O 操作。

ioctl，进行读、写以外的其他操作，参数 cmd 为自定义的命令。

mmap，用于把设备的内容映射到地址空间，一般只有块设备驱动程序使用。

open，打开设备进行 I/O 操作。返回 0 表示成功，返回负数表示失败。如果驱动程序没有提供 open 入口，则只要/dev/device 文件存在就认为打开成功。

release，即 close 操作。

在用户自己的驱动程序中，首先要根据驱动程序的功能，完成 file_operations 结构中函数实现。不需要的函数接口可以直接在 file_operations 结构中初始化为 NULL。file_operations 变量会在驱动程序初始化时，注册到系统内部。当操作系统对设备进行操作时，会调用驱动程序注册的 file_operations 结构中的函数指针。

6.3.1.2 Linux 对中断的处理

在 Linux 系统中，对中断的处理是属于系统核心部分，因而如果设备与系统之间以中断方式进行数据交换，就必须把该设备的驱动程序作为系统核心的一部分。设备驱动程序通过调用 request_irq 函数来申请中断，通过 free_irq 来释放中断，它们被定义为：

```
#include <linux/sched.h>
int request_irq(unsigned int irq,
    void (*handler)(int irq, void dev_id, struct pt_regs *regs);
    unsigned long flags,
    const char *device,
    void *dev_id);
void free_irq(unsigned int irq, void *dev_id);
```

参数 irq 表示所要申请的硬件中断号；handler 为向系统登记的中断处理子程序，中断产生时由系统来调用，调用时所带参数 irq 为中断号；dev_id 为申请时告诉系统的设备标识；regs 为中断产生时的寄存器内容；device 为设备名，将会出现在/proc/interrupts 文件里；flag 是申请时的选项，它决定中断处理程序的一些特性，其中最重要的是中断处理程序是快速处理程序还是慢速处理程序。快速处理程序运行时，所有中断都被屏蔽，而慢速处理程序运行时，除了正在运行的中断外，其他中断都没有被屏蔽。在 Linux 系统中，中断可以被不同的中断处理程序共享。

作为系统核心的一部分，设备驱动程序在申请和释放内存时不是调用 malloc 和 free，而是 kmalloc 和 kfree，它们被定义为：

```
#include <linux/kernel.h>
void *kmalloc(unsigned int len, int priority);
void kfree(void *obj);
```

参数 len 为希望申请的字节数；obj 为要释放的内存指针；priority 为分配内存操作的优先级，即在没有空闲内存时如何操作，一般用 GFP_KERNEL。与中断和内存不同，使用一个没有申请的 I/O 端口不会使系统产生异常，也就不会导致诸如"segmentation fault"一类的错误发生。任何进程都可以访问任何一个 I/O 端口，此时系统无法保证对 I/O 端口的操作不会发生冲突，甚至因此而使系统崩溃，因此，在使用 I/O 端口前，也应该检查此 I/O 端口是否已经有别的程序在使用。若没有，再把此端口标识为正在使用，在使用完以后释放它。

在设备驱动程序中，可以调用 printk 来打印一些调试信息，用法与 printf 类似。Printf 打

印的信息不仅出现在屏幕上，同时还记录在文件 syslog 里。

6.3.1.3 设备驱动的初始化

设备驱动程序所提供的入口点，在设备驱动程序初始化时向系统进行登记，以便系统在适当的时候调用。Linux 系统里，通过调用 register_chrdev 向系统注册字符型设备驱动程序。register_chrdev 定义为：

```
#include <linux/fs.h>
#include <linux/errno.h>
int register_chrdev(unsigned int major, const char *name, struct file_operations *fops);
```

其中，major 是为设备驱动程序向系统申请的主设备号，如果为 0，则系统为此驱动程序动态分配一个主设备号。Name 是设备名。Fops 即上述对各个调用的入口点说明。此函数返回 0 时表示成功。返回-EINVAL 表示申请的主设备号非法，一般来说是主设备号大于系统所允许的最大设备号。返回-EBUSY 表示所申请的主设备号正在被其他设备程序使用。如果动态分配主设备号成功，此函数将返回所分配的主设备号。如果 register_chrdev 操作成功，设备名就会出现在/proc/dvices 文件中。

Linux 为每个设备在/dev 目录中建立一个文件，若用 ls –l 命令列出函数返回值，则小于 0 表示注册失败；返回 0 或者大于 0 的值表示注册成功。

Linux kernel 2.0 支持 128 个主设备号 Linux kernel 2.2 和 2.4 支持 256 个主设备号(0 和 255 保留)。注册以后，Linux 把设备名和主/次设备号联系起来。当有对此设备名的访问时，Linux 通过请求访问的设备名得到主/次设备号，然后把此访问分发到对应的设备驱动，设备驱动再根据次设备号调用不同的函数。

当设备驱动模块从 Linux 内核中卸载，对应的主设备号必须被释放。在模块卸载调用 cleanup_module()函数时，应该调用下面的函数卸载设备驱动：

```
int unregister_chrdev(unsigned int major, const char *name);
```

此函数的参数为主设备号 major 和设备名 name。Linux 内核把 name 和 major 在内核注册的名称对比，如果不相等，卸载失败，并返回-EINVAL；如果 major 大于最大的设备号，也返回-EINVAL。

初始化部分一般还负责给设备驱动程序申请系统资源，包括内存、中断、时钟、I/O 端口等，这些资源也可以在 open 子程序或者其他地方申请。这些资源不用时，应该释放，以利于资源的共享。

设备驱动的初始化函数只要完成的功能是：

(1) 对驱动程序管理的硬件进行必要的初始化

对硬件寄存器进行设置。比如设置中断掩码，设置串口的工作方式、并口的数据方向等。

(2) 初始化设备驱动相关的参数

一般说来，每个设备都要定义一个设备变量，用以保存设备相关的参数。在这里可以对设置变量中的项进行初始化。

(3) 在内核注册设备

Linux 内核通过设备的主设备号和从设备号来访问设备驱动，每个驱动程序都有唯一的主设备号。设备号可以自动获取，内核会分配一个独一无二的主设备号，但这样每次获得

的主设备号可能不一样，设备文件必须重新建立，所有最好手工给设备分配一个主设备号。可以查看 Linux 文件系统中/proc 下的 devices 文件，该文件记录内核中已经使用的主设备号和相应的设备名，选择一个没有被使用的主设备号，调用下面的函数来注册设备：

```
int register_chrdev(unsigned int, const char *, struct filr_operations *)
```

其中三个参数分别表示主设备号、设备名称和上面定义的 filr_operation 结构地址。该函数是在/linux/include/linux/fs.h 中定义的。

(4) 注册中断

如果设备需要 IRQ 支持，则要注册中断。注册中断使用函数：

```
in request_irq(unsigned int irq,
               void (*handler)(int, void *, struct pt_regs *),
               unsigned long flags,
               const char *device,
               void *dev_id);
```

(5) 其他初始化工作

比如给设备分配 I/O。申请 DMA 通道等。

若驱动程序是内核的一部分，则要按如下方式：

```
int __init chr_driver_init(void);
```

声明，注意不能缺少__init。在系统启动时会由内核调用 chr_driver_init，完成驱动程序的初始化。

当驱动程序是以模块的形式编写时，则要按照如下方式：

```
int init_module(void)
```

注：当运行 insmod 命令插入模块时，会调用 init_module 函数完成初始化工作。

6.3.2 设备驱动程序的开发流程

进行嵌入式 Linux 系统的开发，很大的工作量是为各种设备编写驱动程序。在 ARM 平台上开发嵌入式 Linux 的设备驱动程序与在其他平台上开发是一样的。总的来说，实现一个嵌入式 Linux 设备驱动的大致流程如下：

(1) 查看原理图，理解设备的工作原理
(2) 定义主设备号
(3) 在驱动程序中实现驱动的初始化。如果驱动程序采用模块的方式，则要实现模块初始化。
(4) 设计所要实现的文件操作，定义 file_operations 结构。
(5) 实现中断服务(中断并不是每个设备驱动所必须的)
(6) 编译该驱动程序到内核中，或者用 insmod 命令加载
(7) 测试该设备

6.3.3 LED 驱动

SBC-2410X 的 led 设备驱动程序位置：**/friendly-arm/kernel/drivers/char/matrix5-leds.c**

(1) LED 的原理图

SBC-2410X 带有 4 个用户可编程 I/O 方式 LED，如图所示 LED 硬件原理图，下表为 LED 对应的 I/O 口。

表 **6-1** 用户指示灯占用 **CPU** 资源列表

序号	名字	CPU 端口复用资源
1	LED1	nXDREQ1/GPB7
2	LED2	nXDREQ0/GPB8
3	LED3	nXDACK1/GPB9
4	LED4	nXDACK0/GPB10

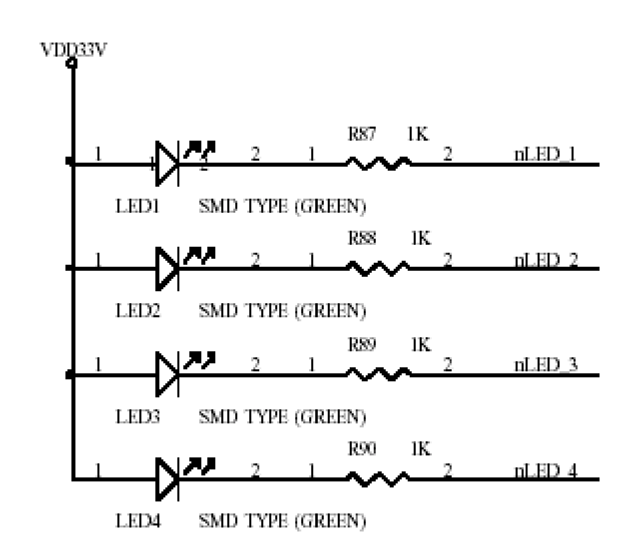


图 **6-6 LED** 原理图

(2) LED 驱动源代码及说明

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

#include <linux/miscdevice.h>
#include <linux/sched.h>
#include <linux/delay.h>
```

```
#include <linux/poll.h>
#include <linux/spinlock.h>
#include <linux/irq.h>
#include <linux/delay.h>

#include <asm/hardware.h>

#define DEVICE_NAME      "leds"    /*定义 led 设备的名字*/
#define LED_MAJOR 231              /*定义 led 设备的主设备号*/

static unsigned long led_table [] = {     /*I/O 方式 led 设备对应的硬件资源*/
 GPIO_B7,
 GPIO_B8,
 GPIO_B9,
 GPIO_B10,
};

/*使用 ioctl 控制 led*/
static int matrix4_leds_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long
arg)
{
 switch(cmd) {
 case 0:
 case 1:
        if (arg > 4) {
               return -EINVAL;
        }
        write_gpio_bit(led_table[arg], !cmd);
 default:
        return -EINVAL;
 }
}
static struct file_operations matrix4_leds_fops = {
 owner:    THIS_MODULE,
 ioctl:    matrix4_leds_ioctl,
};

static devfs_handle_t devfs_handle;
```

```
static int __init matrix4_leds_init(void)
{
 int ret;
 int i;

/*在内核中注册设备*/
 ret = register_chrdev(LED_MAJOR, DEVICE_NAME, &matrix4_leds_fops);
 if (ret < 0) {
    printk(DEVICE_NAME " can't register major number\n");
    return ret;
 }
 devfs_handle = devfs_register(NULL, DEVICE_NAME, DEVFS_FL_DEFAULT,
                 LED_MAJOR, 0, S_IFCHR | S_IRUSR | S_IWUSR, &matrix4_leds_fops,
NULL);

/*使用宏进行端口初始化，set_gpio_ctrl 和 write_gpio_bit 均为宏定义*/
 for (i = 0; i < 8; i++) {
       set_gpio_ctrl (led_table[i] | GPIO_PULLUP_EN | GPIO_MODE_OUT);
       write_gpio_bit(led_table[i], 1);
 }

 printk(DEVICE_NAME " initialized\n");
 return 0;
}

static void __exit matrix4_leds_exit(void)
{
 devfs_unregister(devfs_handle);
 unregister_chrdev(LED_MAJOR, DEVICE_NAME);
}

module_init(matrix4_leds_init);
module_exit(matrix4_leds_exit);
```

(3) 编译安装 LED 驱动

使用手工输入命令的方式编译 led 驱动模块：

#arm-linux-gcc -D__KERNEL__ -I/friendly-arm/kernel/include -DKBUILD_BASENAME=matrix4-leds -DMODULE -c -o matrix5-leds.o matrix5-leds.c

以上命令将生成 matrix5-leds.o 文件，把该文件复制到板子的/lib 目录下，使用以下命令安

装 leds 模块：

#insmod /lib/matrix5-leds.o

删除该模块的命令是：

#rmmod matrix5-leds

注：SBC-2410X 预装的 Linux 内核已经含有该驱动，无法再次安装同样设备名和主设备号的驱动，用户可以自行更改为另外的设备名和主设备号来测试

(4) 测试 LED 驱动

测试 led 的应用程序位于/friendly-arm/examples/leds/目录下，可参考前面的 led 控制。

6.3.4 键盘驱动程序

SBC-2410X 的按键设备驱动程序位置：**/friendly-arm/kernel/drivers/char/matrix5-buttons.c**

(1) 按键原理图

SBC-2410X 带有 4 个用户可编程 I/O 方式按键，如图所示为按键硬件原理图和对应的 CPU 资源。

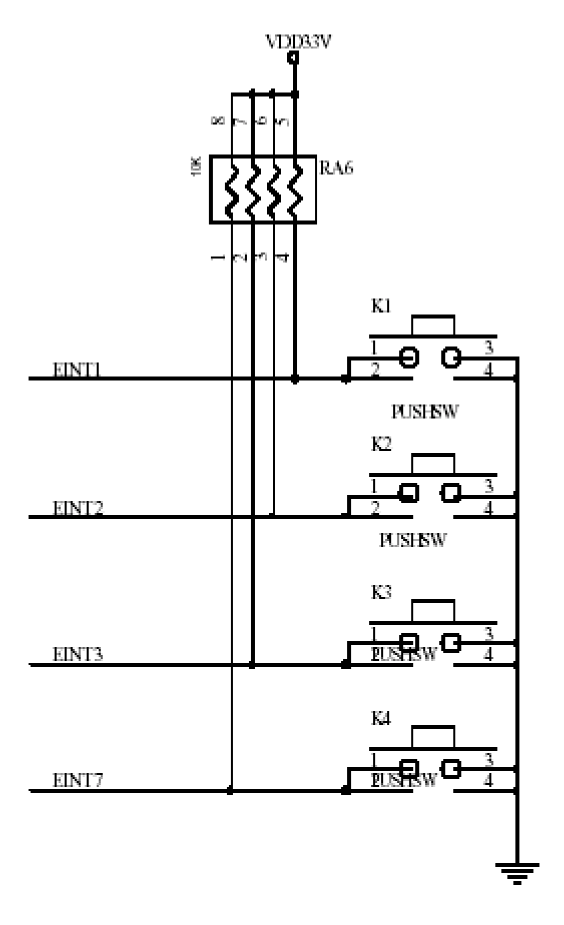


图 **6-7** 按键原理图

(2) 按键设备驱动程序源代码

```
#include <linux/config.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

#include <linux/miscdevice.h>
#include <linux/sched.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/spinlock.h>
#include <linux/irq.h>
#include <linux/delay.h>
```

```
#include <asm/hardware.h>

#define DEVICE_NAME      "buttons" /*定义按键设备名*/
#define BUTTON_MAJOR 232         /*定义按键主设备号*/

static struct key_info {
 int irq_no;
 unsigned int gpio_port;
 int key_no;
} key_info_tab[4] = {                  /*按键所使用的 CPU 资源*/
 { IRQ_EINT1, GPIO_F1, 1 },
 { IRQ_EINT2, GPIO_F2, 2 },
 { IRQ_EINT3, GPIO_F3, 3 },
 { IRQ_EINT7, GPIO_F7, 4 },
};

static int ready = 0;
static int key_value = 0;

static DECLARE_WAIT_QUEUE_HEAD(buttons_wait);

/*按键的中断服务程序*/
static void buttons_irq(int irq, void *dev_id, struct pt_regs *reg)
{
 struct key_info *k;
 int i;
 int found = 0;
 int up;
 int flags;
 for (i = 0; i < sizeof key_info_tab / sizeof key_info_tab[1]; i++) {
        k = key_info_tab + i;
        if (k->irq_no == irq) {
               found = 1;
               break;
        }
 }
 if (!found) {
        printk("bad irq %d in button\n", irq);
```

```
        return;
    }

    save_flags(flags);
    cli();
    set_gpio_mode_user(k->gpio_port, GPIO_MODE_IN);
    up = read_gpio_bit(k->gpio_port);
    set_external_irq(k->irq_no, EXT_BOTH_EDGES, GPIO_PULLUP_DIS);
    restore_flags(flags);
            if (up) {
                    key_value = k->key_no + 0x80;
            } else {
                    key_value = k->key_no;
            }
            ready = 1;
    wake_up_interruptible(&buttons_wait);
   }

   /*申请系统中断，中断方式为双边触发，即在上升沿和下降沿均发生中断*/
   static int request_irqs(void)
   {
    struct key_info *k;
    int i;
    for (i = 0; i < sizeof key_info_tab / sizeof key_info_tab[1]; i++) {
        k = key_info_tab + i;
        set_external_irq(k->irq_no, EXT_BOTH_EDGES, GPIO_PULLUP_DIS);
        if (request_irq(k->irq_no, &buttons_irq, SA_INTERRUPT, DEVICE_NAME,
&buttons_irq)) {
            return -1;
        }

    }
    return 0;
   }

   /*释放中断*/
   static void free_irqs(void)
   {
```

```
 struct key_info *k;
 int i;
 for (i = 0; i < sizeof key_info_tab / sizeof key_info_tab[1]; i++) {
        k = key_info_tab + i;
        free_irq(k->irq_no, buttons_irq);
 }
}

/*file_operations 的“读“指针函数实现*/
static int matrix4_buttons_read(struct file * file, char * buffer, size_t count, loff_t *ppos)
{
 static int key;
 int flags;
 int repeat;
            if (!ready)
                        return -EAGAIN;
            if (count != sizeof key_value)
                        return -EINVAL;
 save_flags(flags);
 if (key != key_value) {
        key = key_value;
        repeat = 0;
 } else {
        repeat = 1;
 }
 restore_flags(flags);

 if (repeat) {
        return -EAGAIN;
 }
             /*使用 copy_to_user 把键值送到用户空间*/
            copy_to_user(buffer, &key, sizeof key);
            ready = 0;
            return sizeof key_value;
}

static unsigned int matrix4_buttons_select(
            struct file *file,
```

```
                struct poll_table_struct *wait)
{
        if (ready)
                return 1;
        poll_wait(file, &buttons_wait, wait);
        return 0;
}


static int matrix4_buttons_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
 switch(cmd) {
 default:
        return -EINVAL;
 }
}

static struct file_operations matrix4_buttons_fops = {
 owner:    THIS_MODULE,
 ioctl:    matrix4_buttons_ioctl,
 poll: matrix4_buttons_select,
 read: matrix4_buttons_read,
};

static devfs_handle_t devfs_handle;
/*按键初始化*/
static int __init matrix4_buttons_init(void)
{
 int ret;

 ready = 0;
/*注册按键设备*/
 ret = register_chrdev(BUTTON_MAJOR, DEVICE_NAME, &matrix4_buttons_fops);
 if (ret < 0) {
    printk(DEVICE_NAME " can't register major number\n");
    return ret;
 }
```

```
        ret = request_irqs();
        if (ret) {
                unregister_chrdev(BUTTON_MAJOR, DEVICE_NAME);
                printk(DEVICE_NAME " can't request irqs\n");
                return ret;
        }
        devfs_handle = devfs_register(NULL, DEVICE_NAME, DEVFS_FL_DEFAULT,
                        BUTTON_MAJOR, 0, S_IFCHR | S_IRUSR | S_IWUSR,
&matrix4_buttons_fops, NULL);

        return 0;
}

static void __exit matrix4_buttons_exit(void)
{
        devfs_unregister(devfs_handle);
        free_irqs();
        unregister_chrdev(BUTTON_MAJOR, DEVICE_NAME);
}

module_init(matrix4_buttons_init);
module_exit(matrix4_buttons_exit);
MODULE_LICENSE("GPL");
```

(3) 编译和安装按键设备驱动模块

使用手工输入命令的方式编译 led 驱动模块：

#arm-linux-gcc -D__KERNEL__ -I/friendly-arm/kernel/include -DKBUILD_BASENAME=matrix4-leds -DMODULE -c -o matrix5-buttons.o matrix5-buttons.c

以上命令将生成 matrix5-buttons.o 文件，把把该文件复制到板子的/lib 目录下，使用以下命令安装 leds 模块：

#insmod /lib/matrix5-buttons.o

删除该模块的命令是：

#rmmod matrix5-buttons

注：SBC-2410X 预装的 Linux 内核已经含有该驱动，无法再次安装同样设备名和主设备号的驱动，用户可以自行更改为另外的设备名和主设备号来测试

(4) 测试按键

测试按键的应用程序位于/friendly-arm/examples/buttons/目录下，可参考前面的按键测试应

用程序编程。

6.3.5 音频设备驱动

SBC-2410X 的声卡设备驱动程序位置：**/friendly-arm/kernel/drivers/sound/s3c2410-uda1341.c**

(1) 基于 I2S 接口的音频系统

S3C2410X 内置的 I2S 总线接口能够和其他厂商提供的多媒体编解码芯片配合使用。提供 I2S 接口能够读取 I2S 总线上面的数据，同时也为 FIFO 数据提供 DMA 的传输模式，这样能够同时传送和接收数据。

S3C2410X 中，有两条串行数据线，一条是输入信号数据线，一条是输出信号数据线，以同时发送和接收数据。I2S 接口有 3 种工作方式：

正常传输模式

正常模式下使用 IISCON 寄存器对 FIFO 进行控制。如果传输 FIFO 缓存为空，IISCON 的第七位被设置成“0”，表示不能继续传输数据，需要 CPU 对缓存进行处理。如果传输 FIFO 缓存非空，IISCON 的第七位被设置成“1”，表示可以继续传输数据。同样，数据接收时，如果 FIFO 满，标识位是“0”，此时需要 CPU 对 FIFO 进行处理，如果 FIFO 没有满，那么标志位是“1”，这个时候可以继续接收数据。

DMA 模式

通过设置 IISFCON 寄存器可以使 IIS 接口工作在这种模式下。这种模式，FIFO 寄存器组的控制权掌握在 DMA 控制器上，当 FIFO 满了，由 DMA 控制器对 FIFO 中的数据进行处理。DMA 模式的选择由 IISCON 寄存器的第 4 和第 5 位控制。

传输/接受模式

这种模式下，I2S 数据可以同时接收和发送音频数据。

下图是 SBC-2410X 与飞利浦公司的 UDA1341TS 芯片的连接图。

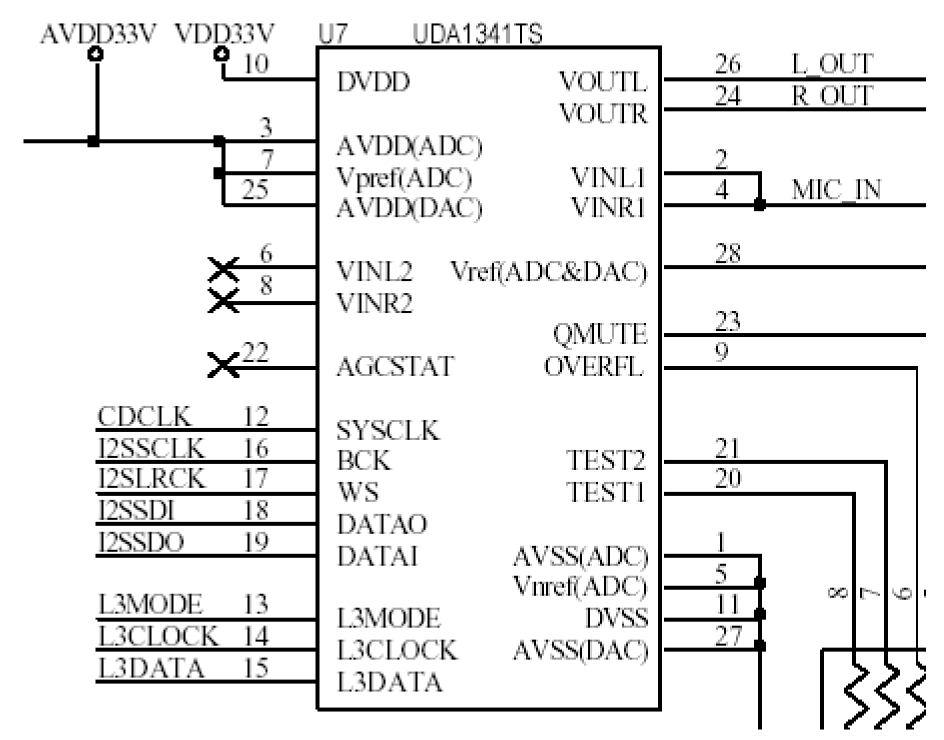


图 6-8 SBC-2410X 中音频系统原理图

如图所示，S3C2410X 的 I2S 总线时钟信号 SCK 与 UDA1341TS 的 BCK 连接，字段选择连接在 WS 引脚上。UDA1341TS 提供了两个音频通道，分别用于输入和输出，对应的引脚连接为：I2S 总线的音频输出 I2SSDO 对应于 UDA1341 的音频输入；I2S 总线的音频输入 I2SSDI 对应于 UDA1341 的音频输出。UDA1341TS 的 L3 接口相当于一个混音器控制接口，可以用来控制输入/输出音频信号的音量大小。低音等。L3 接口的引脚 L3MODE，L3DATA，L3CLOCK 分别连接到 S3C2410 的 GPB2、GPOB3、GPB4、三个通用数据输出引脚上。

应用程序第一次使用音频设备，并向设备传送数值的时候，驱动程序必须完成下面的任务：

①通过程序控制音频设备，并且为设备准备好工作参数(包括速度、声道、采样宽度)。同时设置好对应的传输总线 I2S 和 L3。

②根据采样参数计算出缓冲段的大小(程序也可以指定缓冲区的大小)，分配对应的 DMA 空间供设备使用。

③向缓冲区填入应用程序生成的数据。

④如果第一个缓冲区填充完毕，音频设备就开始播放填入的数据，对于播放要求比较高的应用，可以等两个或者更多的缓冲区段被填充满以后再开始播放。

⑤应用程序继续填充缓冲区，如果所有的缓冲区都被使用，那么应用程序将转入挂起状态，直到第一个缓存区内的数据播放完毕，音频设备发出一个中断，通知应用应用程序继续向缓存区输入数据。

以后的操作过程和上面的过程类似，只不过应用程序不用初始化音频设备和缓冲区。

应用程序从设备里读取数据的操作和向设备里写数据的操作是类似的，具体过程如下：

①通过程序控制音频设备，并且为设备设置好工作参数(包括速度、声道、采样宽度)。

②根据采样参数计算出缓冲段的大小(程序也可以指定缓冲区的大小)

③激活录音设备开始录音。

④录音设备将处理好的数据填入缓冲区，在这个期间应用程序将被挂起，直到缓冲区被填满。

⑤当第一个缓冲区被填满后，应用程序将缓冲区的数据拷贝到应用程序的内存区域。

⑥当缓冲区中的数据读完之后，读操作挂起，等待音频设备填充其他的缓存区，直到录音结束。以后从音频设备读取数据的操作过程和上面的过程相似，只不过应用程序不用初始化音频设备的缓冲区。

当应用程序处理区内数据的速度和音频设备产生的数据的速度不匹配时，就会出现数据丢失的现象。比如录音的时候，操作时实时的，当音频设备填充完所有的缓存区，而应用程序却没有处理完缓冲区的数据时，那么新录制的声音数据将会丢失，播放音频数据的时候也会出现这样的问题。

(2) 音频设备驱动程序

UDA1341 的驱动程序位于：/friendly-arm/kernel/drivers/sound/s3c2410-uda1341.c

该设备驱动代码比较多，下面以以下几个主要模块分别介绍：

①UDA1341 驱动的初始化

②打开音频设备

③缓存区设计

④ioctl 系统调用

⑤音频数据的播放和录制

①UDA1341 驱动的初始化

```
int __init s3c2410_uda1341_init(void)
{
 unsigned long flags;

 local_irq_save(flags);


/*CPU 控制端口初始化*/
 /* GPB 4: L3CLOCK, OUTPUT */
 set_gpio_ctrl(GPIO_L3CLOCK);
 /* GPB 3: L3DATA, OUTPUT */
 set_gpio_ctrl(GPIO_L3DATA);
 /* GPB 2: L3MODE, OUTPUT */
 set_gpio_ctrl(GPIO_L3MODE);
```

```
    /* GPE 3: I2SSDI */
    set_gpio_ctrl(GPIO_E3 | GPIO_PULLUP_EN | GPIO_MODE_I2SSDI);
    /* GPE 0: I2SLRCK */
    set_gpio_ctrl(GPIO_E0 | GPIO_PULLUP_EN | GPIO_MODE_I2SSDI);
    /* GPE 1: I2SSCLK */
    set_gpio_ctrl(GPIO_E1 | GPIO_PULLUP_EN | GPIO_MODE_I2SSCLK);
    /* GPE 2: CDCLK */
    set_gpio_ctrl(GPIO_E2 | GPIO_PULLUP_EN | GPIO_MODE_CDCLK);
    /* GPE 4: I2SSDO */
    set_gpio_ctrl(GPIO_E4 | GPIO_PULLUP_EN | GPIO_MODE_I2SSDO);

    local_irq_restore(flags);

   /*初始化 UDA1341*/
    init_uda1341();

    output_stream.dma_ch = DMA_CH2;

    if (audio_init_dma(&output_stream, "UDA1341 out")) {
        audio_clear_dma(&output_stream);
        printk( KERN_WARNING AUDIO_NAME_VERBOSE
            ": unable to get DMA channels\n" );
        return -EBUSY;
    }

    input_stream.dma_ch = DMA_CH1;

        if (audio_init_dma(&input_stream, "UDA1341 in")) {
            audio_clear_dma(&input_stream);
            printk( KERN_WARNING AUDIO_NAME_VERBOSE
                ": unable to get DMA channels\n" );
            return -EBUSY;
        }

   /*注册音频设备*/
    audio_dev_dsp = register_sound_dsp(&smdk2410_audio_fops, -1);
    audio_dev_mixer = register_sound_mixer(&smdk2410_mixer_fops, -1);
```

```
 printk(AUDIO_NAME_VERBOSE " initialized\n");

 return 0;
}
```

②打开音频设备

```
static int smdk2410_audio_open(struct inode *inode, struct file *file)
{
 int cold = !audio_active;

 DPRINTK("audio_open\n");

 if ((file->f_flags & O_ACCMODE) == O_RDONLY) {
        if (audio_rd_refcount || audio_wr_refcount)
               return -EBUSY;
        audio_rd_refcount++;
 } else if ((file->f_flags & O_ACCMODE) == O_WRONLY) {
        if (audio_wr_refcount)
               return -EBUSY;
        audio_wr_refcount++;
 } else if ((file->f_flags & O_ACCMODE) == O_RDWR) {
        if (audio_rd_refcount || audio_wr_refcount)
               return -EBUSY;
        audio_rd_refcount++;
        audio_wr_refcount++;
 } else
        return -EINVAL;

 if (cold) {
        audio_rate = AUDIO_RATE_DEFAULT;
        audio_channels = AUDIO_CHANNELS_DEFAULT;
        audio_fragsize = AUDIO_FRAGSIZE_DEFAULT;
        audio_nbfrags = AUDIO_NBFRAGS_DEFAULT;
        if ((file->f_mode & FMODE_WRITE)){
                      init_s3c2410_iis_bus_tx();
                      audio_clear_buf(&output_stream);
        }
        if ((file->f_mode & FMODE_READ)){
```

```
                init_s3c2410_iis_bus_rx();
                audio_clear_buf(&input_stream);
        }
 }

 MOD_INC_USE_COUNT;

 return 0;
}
```

③缓存区设计

嵌入式系统为了提高系统的吞吐量，经常使用 DMA 技术直接将需要回放或者是录制的声音存放在内核的 DMA 缓存区中，DMA 缓存区可以是 ARM 芯片上的 RAM，也可以是片外的 RAM。应用程序使用 read 或 write 系统调用在内核缓冲和应用程序缓冲中交换数据。

为了解决音频应用 I/O 数据量大的问题，最简单易行的解决方法是使用比较大的缓存区，但是比较大的缓存区在使用的时候会出现延时。为了解决延时问题，现在大多数的 Linux 音频设备的驱动程序都使用了多段缓存机制，这种机制将一个大的缓存区分割成若干个相同的大小的块(这些块也被称为“段”)。这样对较大的缓存区的操作就转变给对较小的缓冲段的操作，于是可以在不增加对缓存区操作时间的情况下增加缓存区的大小。应用程序层面上，块的大小和个数可以使用 ioctl 系统调用来调整。

一般说来块的大小取决于以下 4 个方面：

音频设备的计算能力

音频的质量

应用程序的性质

处理器的能力和处理器的调试方式

三星的DMA控制器没有内置的DMA存储区域，所以驱动程序中必须为音频设备分配DMA缓存区，这样就能通过 DMA 直接将需要的回放或者录音数据放在内核的 DMA 缓存区中。此外为了发挥 DMA 的能力，驱动程序中必须合理的设计缓存区。

音频设备的 DMA 缓存区的初始化过程如下：

```
static int audio_setup_buf(audio_stream_t * s)
{
 int frag;
 int dmasize = 0;
 char *dmabuf = 0;
 dma_addr_t dmaphys = 0;

 if (s->buffers)
        return -EBUSY;
```

```
/*为缓存分配内存*/
 s->nbfrags = audio_nbfrags;
 s->fragsize = audio_fragsize;

 s->buffers = (audio_buf_t *)
        kmalloc(sizeof(audio_buf_t) * s->nbfrags, GFP_KERNEL);
 if (!s->buffers)
        goto err;

/*清空内存*/
 memset(s->buffers, 0, sizeof(audio_buf_t) * s->nbfrags);

 for (frag = 0; frag < s->nbfrags; frag++) {
        audio_buf_t *b = &s->buffers[frag];

        /*为 DMA 分配连续的内存空间*/
        if (!dmasize) {
               dmasize = (s->nbfrags - frag) * s->fragsize;
               do {
                      dmabuf = consistent_alloc(GFP_KERNEL|GFP_DMA,
                                          dmasize, &dmaphys);
                      if (!dmabuf)
                                   dmasize -= s->fragsize;
               } while (!dmabuf && dmasize);
               if (!dmabuf)
                      goto err;
               b->master = dmasize;
        }

        /*audio_buf 的初始化*/
        b->start = dmabuf;
        b->dma_addr = dmaphys;
        sema_init(&b->sem, 1);
        DPRINTK("buf %d: start %p dma %d\n", frag, b->start, b->dma_addr);

        dmabuf += s->fragsize;
        dmaphys += s->fragsize;
        dmasize -= s->fragsize;
```

```
}

s->buf_idx = 0;
s->buf = &s->buffers[0];

return 0;

	err:
printk(AUDIO_NAME ": unable to allocate audio memory\n ");

/*清空缓存区*/
audio_clear_buf(s);
return -ENOMEM;
}
```

程序中的 for 循环语句，将每个分配好内存空间的缓存区段的地址信息复制到管理缓存区的 audio_buf_t 类型的结构体中。这里在分配内存区域的时候使用了一个技巧，由于分配 DMA 使用的内存空间时需要尽可能地使用连续空间，因此要尽量一次分配和需要地缓存空间大小相同的连续内存页面。For 循环体中 if(!dmasize)条件判断语句仅仅在第一次 for 循环的时候进入循环体进行内存分配。内存分配里面有一个 do-while 类型的循环，循环体第一次尝试的分配大小为“段大小 X 段个数”。如果分配不成功，就分配大小为“段大小 X(段个数-1)”的连续内存空间，还不成功就分配大小“段大小 X(段个数-2)”的连续内存空间，依此类推。

④ioctl 系统调用

Ioctl 可以调整音频设备的缓存区参数，为改善应用程序的音频质量提供了接口。此外 ioctl 也被用来对音频设备做其他控制。整个 ioctl 其实就是一个大的 switch-case 语句，根据不同的输入命令参数调用不同的函数。

当音频芯片连接 I2S 总线上时，必须在驱动里面解决数据 I/O 和 IIS 总线的速度匹配问题。S3C2410X 上的 IIS 控制器，在 8.00khz 的采样频率下，如果主控时钟 codeclk 使用 384 的倍频，那么主时钟频率就是 8x384=3.0720MHz，对于分频器 A 来说，由于 PCLK/(N+1)=3.072MHz，所以 N=50.75/3.072-1=15.67=0x10。IISPSR 寄存器的 0 到 4 比特位存放的是分频控制器 B 的控制信息，5 到 9 比特位存放的是分频控制器 A 的控制信息。所以相关的 IISPSR 急促你去的设置为：

```
rIISPSR=(15<<5)+15;
```

下面是 ioctl 的控制实现代码，其中包含根据不同的采样频率对 IIS 总线的设置：

```
static int smdk2410_audio_ioctl(struct inode *inode, struct file *file,
                                uint cmd, ulong arg)
{
long val;
```

```
switch (cmd) {
	case SNDCTL_DSP_SETFMT:
		get_user(val, (long *) arg);
		if (val & AUDIO_FMT_MASK) {
				audio_fmt = val;
				break;
		} else
			return -EINVAL;

	case SNDCTL_DSP_CHANNELS:
	case SNDCTL_DSP_STEREO:
		get_user(val, (long *) arg);
		if (cmd == SNDCTL_DSP_STEREO)
			val = val ? 2 : 1;
		if (val != 1 && val != 2)
			return -EINVAL;
		audio_channels = val;
		break;

	case SOUND_PCM_READ_CHANNELS:
		put_user(audio_channels, (long *) arg);
		break;

	case SNDCTL_DSP_SPEED:
		get_user(val, (long *) arg);
		val = audio_set_dsp_speed(val);
					if (val < 0)
			return -EINVAL;
		put_user(val, (long *) arg);
		break;

	case SOUND_PCM_READ_RATE:
		put_user(audio_rate, (long *) arg);
		break;

	case SNDCTL_DSP_GETFMTS:
		put_user(AUDIO_FMT_MASK, (long *) arg);
		break;
```

```
case SNDCTL_DSP_GETBLKSIZE:
	if(file->f_mode & FMODE_WRITE)
		return put_user(audio_fragsize, (long *) arg);
	else
		return put_user(audio_fragsize, (int *) arg);

case SNDCTL_DSP_SETFRAGMENT:
		if (file->f_mode & FMODE_WRITE) {
		if (output_stream.buffers)
			return -EBUSY;
		get_user(val, (long *) arg);
		audio_fragsize = 1 << (val & 0xFFFF);
		if (audio_fragsize < 16)
			audio_fragsize = 16;
		if (audio_fragsize > 16384)
			audio_fragsize = 16384;
		audio_nbfrags = (val >> 16) & 0x7FFF;
		if (audio_nbfrags < 2)
			audio_nbfrags = 2;
		if (audio_nbfrags * audio_fragsize > 128 * 1024)
			audio_nbfrags = 128 * 1024 / audio_fragsize;
		if (audio_setup_buf(&output_stream))
			return -ENOMEM;

	}
	if (file->f_mode & FMODE_READ) {
		if (input_stream.buffers)
			return -EBUSY;
		get_user(val, (int *) arg);
		audio_fragsize =    1 << (val & 0xFFFF);
		if (audio_fragsize < 16)
			audio_fragsize = 16;
		if (audio_fragsize > 16384)
								audio_fragsize = 16384;
							audio_nbfrags = (val >> 16) & 0x7FFF;
							if (audio_nbfrags < 2)
								audio_nbfrags = 2;
```

```
                        if (audio_nbfrags * audio_fragsize > 128 * 1024)
                            audio_nbfrags = 128 * 1024 / audio_fragsize;
                        if (audio_setup_buf(&input_stream))
                            return -ENOMEM;

        }
        break;

    case SNDCTL_DSP_SYNC:
        return audio_sync(file);

    case SNDCTL_DSP_GETOSPACE:
    {
        audio_stream_t *s = &output_stream;
        audio_buf_info *inf = (audio_buf_info *) arg;
        int err = verify_area(VERIFY_WRITE, inf, sizeof(*inf));
        int i;
        int frags = 0, bytes = 0;

        if (err)
            return err;
        for (i = 0; i < s->nbfrags; i++) {
            if (atomic_read(&s->buffers[i].sem.count) > 0) {
                if (s->buffers[i].size == 0) frags++;
                bytes += s->fragsize - s->buffers[i].size;
            }
        }
        put_user(frags, &inf->fragments);
        put_user(s->nbfrags, &inf->fragstotal);
        put_user(s->fragsize, &inf->fragsize);
        put_user(bytes, &inf->bytes);
        break;
    }

    case SNDCTL_DSP_GETISPACE:
    {
        audio_stream_t *s = &input_stream;
        audio_buf_info *inf = (audio_buf_info *) arg;
```

```
        int err = verify_area(VERIFY_WRITE, inf, sizeof(*inf));
        int i;
        int frags = 0, bytes = 0;

        if (!(file->f_mode & FMODE_READ))
                        return -EINVAL;

        if (err)
            return err;
        for(i = 0; i < s->nbfrags; i++){
        if (atomic_read(&s->buffers[i].sem.count) > 0)
                        {
                            if (s->buffers[i].size == s->fragsize)
                                frags++;
                            bytes += s->buffers[i].size;
                        }
                    }
        put_user(frags, &inf->fragments);
                    put_user(s->nbfrags, &inf->fragstotal);
                    put_user(s->fragsize, &inf->fragsize);
                    put_user(bytes, &inf->bytes);
                    break;
    }
    case SNDCTL_DSP_RESET:
        if (file->f_mode & FMODE_READ) {
                        audio_clear_buf(&input_stream);
                    }
                    if (file->f_mode & FMODE_WRITE) {
                        audio_clear_buf(&output_stream);
                    }
                    return 0;
    case SNDCTL_DSP_NONBLOCK:
        file->f_flags |= O_NONBLOCK;
                    return 0;
    case SNDCTL_DSP_POST:
        case SNDCTL_DSP_SUBDIVIDE:
        case SNDCTL_DSP_GETCAPS:
        case SNDCTL_DSP_GETTRIGGER:
```

```
                case SNDCTL_DSP_SETTRIGGER:
                case SNDCTL_DSP_GETIPTR:
                case SNDCTL_DSP_GETOPTR:
                case SNDCTL_DSP_MAPINBUF:
                case SNDCTL_DSP_MAPOUTBUF:
                case SNDCTL_DSP_SETSYNCRO:
                case SNDCTL_DSP_SETDUPLEX:
                return -ENOSYS;
        default:
                return smdk2410_mixer_ioctl(inode, file, cmd, arg);
 }

 return 0;
}
```

⑤**音频数据的播放和录制**

音频数据的录制和播放对应驱动程序里面的函数是 write 和 read，当内存和 DMA 的函数已经制作完成后，录音和回放相关的驱动程序制作也就比较简单了。这两个驱动例程的实现比较接近，下面是录音部分的代码及注释：

```
static ssize_t smdk2410_audio_read(struct file *file, char *buffer,
                                                    size_t count, loff_t * ppos)
{
 const char *buffer0 = buffer;
                audio_stream_t *s = &input_stream;
                int chunksize, ret = 0;

                DPRINTK("audio_read: count=%d\n", count);

                if (ppos != &file->f_pos)
                        return -ESPIPE;
/*缓存区检查*/
 if (!s->buffers) {
        int i;

                        if (audio_setup_buf(s))
                                return -ENOMEM;
                        /*依次从缓冲块中读取数据*/
                        for (i = 0; i < s->nbfrags; i++) {
                                audio_buf_t *b = s->buf;
```

```
                down(&b->sem);
                s3c2410_dma_queue_buffer(s->dma_ch, (void *) b,
                                b->dma_addr, s->fragsize, DMA_BUF_RD);
                NEXT_BUF(s, buf);
            }
        }

        while (count > 0) {
            audio_buf_t *b = s->buf;

            /*  等待缓冲区被填满*/
            if (file->f_flags & O_NONBLOCK) {
                ret = -EAGAIN;
                if (down_trylock(&b->sem))
                    break;
            } else {
                ret = -ERESTARTSYS;
                if (down_interruptible(&b->sem))
                    break;
            }

            /*从缓冲区读取数据*/
            chunksize = b->size;
    if (chunksize > count)
                chunksize = count;
            DPRINTK("read %d from %d\n", chunksize, s->buf_idx);

            /*将数据从内核空间拷贝到用户空间*/
            if (copy_to_user(buffer, b->start + s->fragsize - b->size,
                chunksize)) {
                up(&b->sem);
                return -EFAULT;
            }

            b->size -= chunksize;

            buffer += chunksize;
            count -= chunksize;
```

```
                if (b->size > 0) {
                        up(&b->sem);
                        break;
                }

                /*  释放缓存区  */
                s3c2410_dma_queue_buffer(s->dma_ch, (void *) b,
                  b->dma_addr, s->fragsize, DMA_BUF_RD);

                NEXT_BUF(s, buf);
        }

        if ((buffer - buffer0))
                ret = buffer - buffer0;

//      DPRINTK("audio_read: return=%d\n", ret);

        return ret;
}
```

驱动程序的输入参数中，struct file 类型的文件指针是音频设备的设备文件，buffer 参数是用户应用程序内存空间中的缓存区。最要一个参数是偏移量参数，要求设备输入和设备文件中的偏移量参数相同，这个参数一般为 0。

程序先初始化函数的局部变量，然后对输入参数进行检查，再通过 if(s->mapped)语句判断设备文件的内核缓冲区是否被映射到用户空间中，如果被映射了，那么后面的内存拷贝程序就没有必要了。

接下来是对 s->actice 的判断，如果和设备文件相关的缓存区还没有被激活，那么就初始化缓存区，对于复杂的系统，可以使用循环链表作为 DMA 缓存区的数据结构，如果使用的是这样的 DMA 管理程序，那么在初始化缓存区以后还要将缓存区添加到 DMA 缓存区的循环链表中。接下来的 while 循环中使用 copy_to_user()将系统缓存区中的数据拷贝到用户缓存区中，这里使用 copy_to_user()的原因是因为驱动程序使用的是内核空间，而应用程序使用的是用户空间，应用程序不能直接读取内核空间的数据。

拷贝完后使用 dma_queue_buffer()将缓存区释放。

6.4 配置和编译 Bootloader

6.4.1 bootloader 概述

在嵌入式系统中，不像 PC 那样，在主板上有一个 CMOS，用来存放固件(firmware)。而是使用一个被称作 bootloader 的程序，用以启动系统和作简单的管理。

实际上，在一个稍微复杂的嵌入式系统中，bootloader 是十分重要的，它有如下作用：

- 把内核(kernel)从 flash 复制到 RAM，然后启动它
- 初始化硬件
- 下载程序并写入 flash(一般通过串口或者网口先把内核下载到 RAM 中，然后写入到 flash)
- 检测目标板(bootloader 会有一些简单的代码用以测试目标板硬件的好坏)

vivi 是什么？

Vivi 是由 mizi 公司设计为 ARM 处理器系列设计的一个 bootloader，因为 vivi 目前只支持使用串口和主机通信，所以您必须使用一条串口电缆来连接目标板和主机。在 SBC-2410X 中，使用的是一条标准公母串口线。

6.4.2 编译 vivi

首先，进入 vivi 源代码目录：

#cd /friendly-arm/vivi

然后配置和编译它，如下步骤：

执行“**make menuconfig**”命令开始配置内核，实际上，你不需要自己手工选择配置它，已经配置好了，只需装载一个缺省的配置文件(default-configuration-file)即可，使用这个配置文件生成的 vivi 正好适合于目标板，这个配置文件在 vivi/arch/def-configs 目录中，该目录包含了一些适合于各种板的配置文件。

装载“arch/def-configs/SBC-2410X”，然后保存该设置，并执行“make”命令编译 vivi。

#cd /friendly-arm/vivi

#make menuconfig

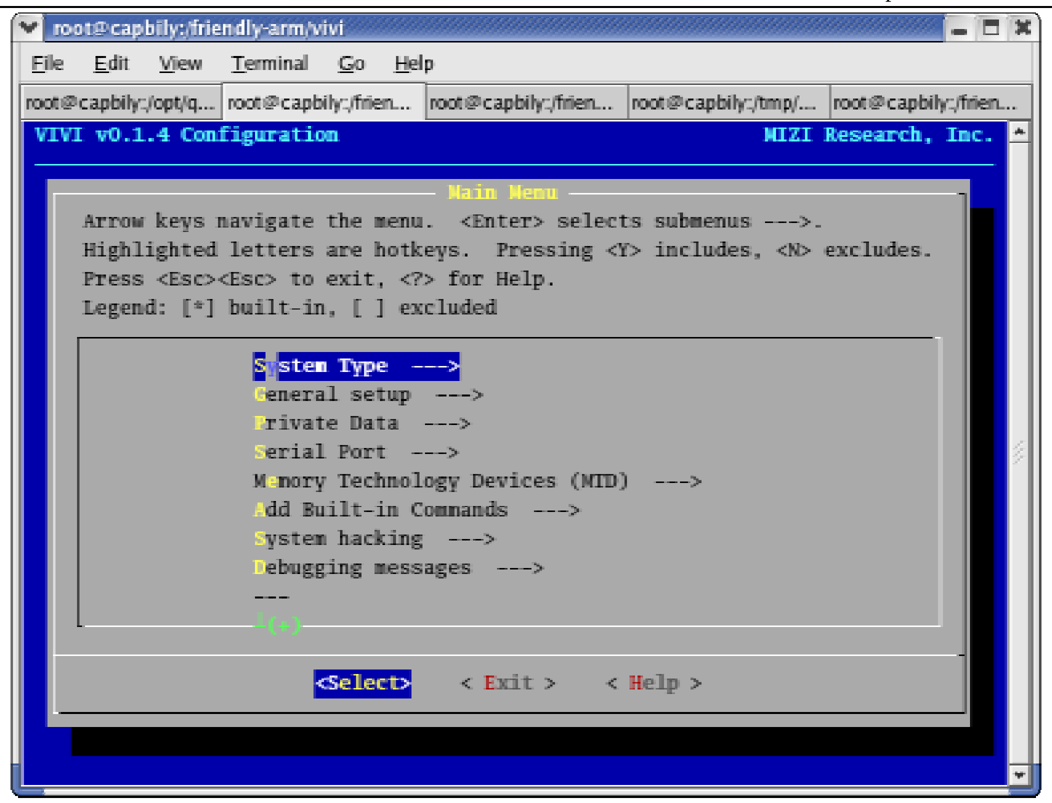


图 6-9 执行"make menuconfig"配置 vivi

在跳出的窗口中选择"Load on Alternate Configuration File"菜单，然后输入"arch/def-configs/SBC-2410X"，如图 6-10-所示。

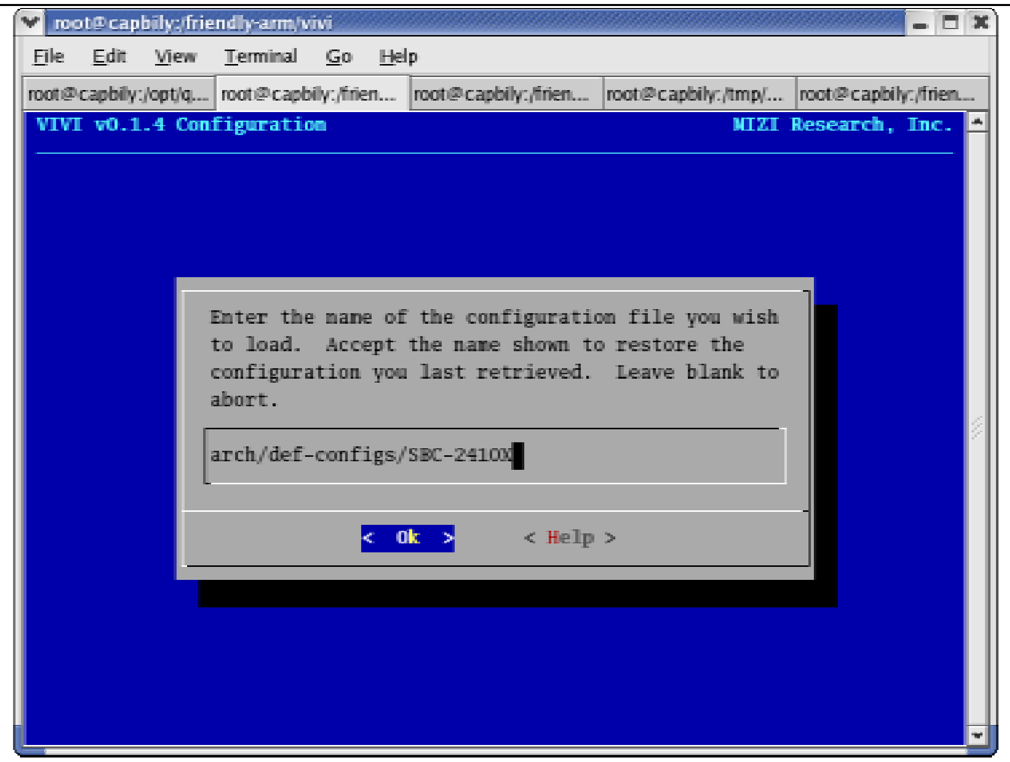


图 6-10 装载缺省配置文件

再选择“OK”，图 6-9 又会出现一次，这时选择退出，跳出如图 6-11 所示界面。选择“Yes”保存刚才的配置。

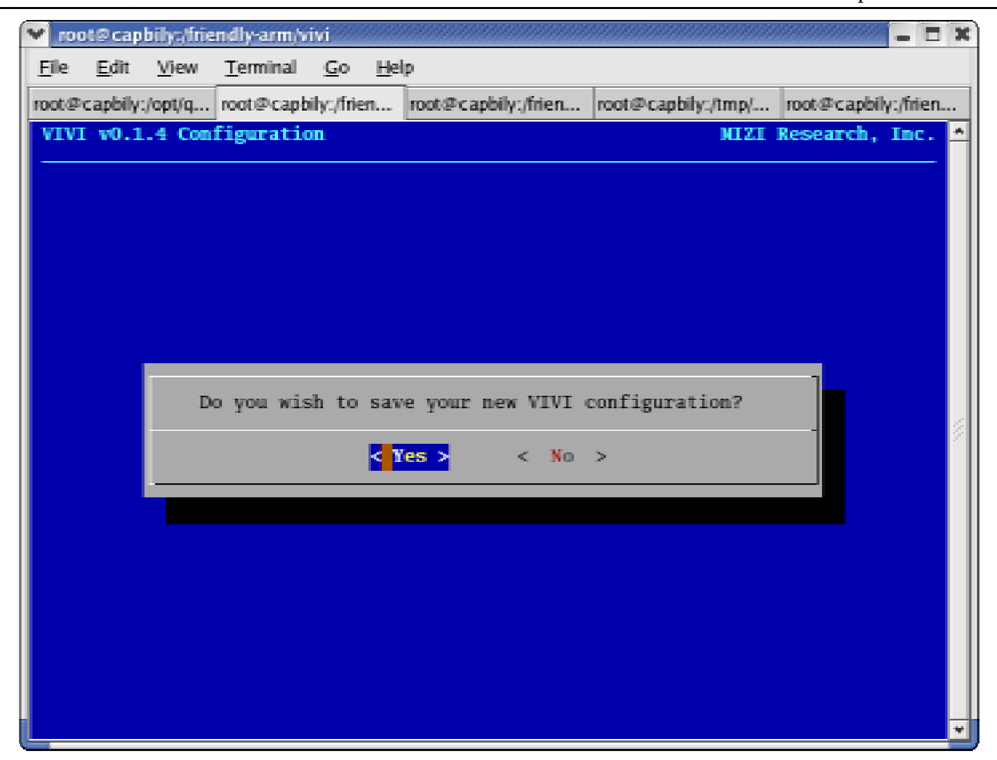


图 6-11 保存配置

保存完毕，执行以下命令：

#make

如果编译过程顺利，将会在当前目录下生成 vivi 二进制映象文件。

6.5 配置和编译内核(kernel)

SBC-2410X 所用的内核源代码(kernel sources)位于/friendly-arm/kernel 目录。像编译 vivi 一样，编译之前要运行"**make menuconfig**"命令配置内核。内核的选项很多，但是我们不需要手工逐个选择，当您将 kernel.tgz 解压到/friendly-arm 目录后，已经缺省配置了 SBC-2410X 所需要的各个组件模块。如果要编译内核您只需要执行 make 即可。更多的内核选项可以参考相关书籍，内容大致相同。

#make zImage

最后生成的内核映像文件 zImageI 位于 arch/arm/boot 目录。

6.6 系统的启动

一个完整的 Linux 系统包括一个 Kernel 和一个 root 文件系统。如果只有一个 Linux 内核，系统最后是无法正确引导的。Linux 内核引导完毕，就试图 mount 一个 root 文件系统(根文件系

统)，如果找不到，内核将报如下错误：

Kenel panic：VFS：Unable to mount root fs on 02:00

这个文件系统的位置由环境变量 root 指定，SBC-2410X 目前支持 3 种启动方式

- root=/dev/bon/2()
- root=/dev/mtdblock/0 本地启动(缺省启动方式)
- root=/dev/nfs

其中第一种 root=/dev/bon/2 是 mizi 公司提供的启动方法，它使用只读文件系统 cramfs 作为根系统。基于该文件系统的所有文件操作均是只读的，用户只能在内存中创建文件，系统掉电后，文件也就不存在了，目前 **SBC-2410X 已经摒弃了这种只读的 cramfs 系统启动方式。**

yaffs 是一种专门为嵌入式系统中常用的 flash 设备设计的一种可读写的文件系统，它比 jffs2 文件系统具有更快的启动速度，对 Flash 使用寿命有更好的保护机制，SBC-2410X 目前移植了具有商业水准的 yaffs，性能及稳定行都已经过严格的测试。

还可以使用 NFS 通过以太网挂接根文件系统，这是一种经常用来作为调试使用的文件系统启动方式。通过网络挂接的根文件系统，可以在主机上生成 ARM 交叉编译版本的目标文件或二进制可执行文件，然后就可以直接装载或执行它，而不用频繁地写入 flash。

6.6.1 通过 yaffs 启动系统

SBC-2410X 的缺省启动文件系统是 yaffs，系统启动后将在串口打印如下所示信息：

注意：**启动时按下空格键可引入 vivi 提示符模式。**

VIVI version 0.1.4 (root@capbily) (gcc version 2.95.3 20010315 (release)) #0.1.4 Mon Oct 4 16:20:35 CST 2004
MMU table base address = 0x33DFC000
Succeed memory mapping.
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Could not found stored vivi parameters. Use default vivi parameters.
Press Return to start the LINUX now, any other key for vivi
Copy linux kernel from 0x00030000 to 0x30008000, size = 0x00100000 ... done
zImage magic = 0x016f2818
Setup linux parameters at 0x30000100
linux command line is: "noinitrd root=/dev/mtdblock/0 init=/linuxrc console=ttyS0"
MACH_TYPE = 193
NOW, Booting Linux......
Uncompressing Linux.. done, booting the kernel.
Linux version 2.4.18-rmk7-pxa1 (root@capbily) (gcc version 2.95.3 20010315 (release)) #9 Wed Oct 6 17:00:54 CST 2004

```
CPU: ARM/CIRRUS Arm920Tsid(wb) revision 0
Machine: Samsung-SMDK2410
On node 0 totalpages: 16384
zone(0): 16384 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: noinitrd root=/dev/mtdblock/0 init=/linuxrc console=ttyS0
DEBUG: timer count 15626
Calibrating delay loop... 99.94 BogoMIPS
Memory: 64MB = 64MB total
Memory: 62676KB available (1374K code, 364K data, 56K init)
Dentry-cache hash table entries: 8192 (order: 4, 65536 bytes)
Inode-cache hash table entries: 4096 (order: 3, 32768 bytes)
Mount-cache hash table entries: 1024 (order: 1, 8192 bytes)
Buffer-cache hash table entries: 4096 (order: 2, 16384 bytes)
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
CPU clock = 200.000 Mhz, HCLK = 100.000 Mhz, PCLK = 50.000 Mhz
Initializing S3C2410 buffer pool for DMA workaround
usbctl: zombie --> [reset] --> default. Device in default state.
S3C2410 USB Controller Core Initialized
USB Function Character Driver Interface - 0.5, (C) 2001, Extenex Corp.
usbctl: Opened for usb-char
usbctl: Started for usb-char
Starting kswapd
devfs: v1.10 (20020120) Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x1
ttyS%d0 at I/O 0x50000000 (irq = 52) is a S3C2410
ttyS%d1 at I/O 0x50004000 (irq = 55) is a S3C2410
ttyS%d2 at I/O 0x50008000 (irq = 58) is a S3C2410
pty: 256 Unix98 ptys configured
leds initialized
S3C2410 Real Time Clock Driver v0.1
block: 128 slots per queue, batch=32
Uniform Multi-Platform E-IDE driver Revision: 6.31
```

ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
eth0: cs8900 rev J(3.3 Volts) found at 0xd0000300
cs89x0 media RJ-45, IRQ 37
Linux video capture interface: v1.00
SCSI subsystem driver Revision: 1.00
scsi0 : SCSI host adapter emulation for IDE ATAPI devices
UDA1341 audio driver initialized
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
bon0: 00000000-0002c000 (0002c000) 00000000
bon1: 00030000-00130000 (00100000) 00000000
bon2: 00130000-03ffc000 (03ecc000) 00000000
usb.c: registered new driver usbdevfs
usb.c: registered new driver hub
usb-ohci.c: USB OHCI at membase 0xe9000000, IRQ 26
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
port #1 suspened!
port #0 alived!
hub.c: 1 port detected
usb.c: registered new driver usblp
printer.c: v0.8:USB Printer Device Class driver
usb.c: registered new driver ov511
ov511.c: v1.48a for Linux 2.4 : OV511 USB Camera Driver
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
USB Mass Storage support registered.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 4096 bind 4096)
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
FAT: bogus logical sector size 42986
yaffs: dev is 7936 name is "1f:00"
VFS: Mounted root (yaffs filesystem).
Mounted devfs on /dev
Freeing init memory: 56K
==

```
=          Friendly-ARM Tech. Ltd.                =
=            http://www.arm9.net                  =
=          http://www.arm9.com.cn                 =
=========================================

init started:   BusyBox v0.60.5 (2003.09.05-09:25+0000) multi-c 鸭 l binary
Using /lib/input.o
Using /lib/keybdev.o
insmod: unresolved symbol handle_scancode
insmod: unresolved symbol keyboard_tasklet
insmod: unresolved symbol kbd_ledfunc
Using /lib/mousedev.o
mice: PS/2 mouse device common for all mice
Using /lib/evdev.o
Using /lib/lcd640x480x32.o
lcd address is c4950000(33800000)
LCD640x480x32 installed, by capbily@hotmail.com(http://www.arm9.net
Using /lib/hid-core.o
insmod: unresolved symbol hidinput_connect
insmod: unresolved symbol hidinput_hid_event
insmod: unresolved symbol hidinput_disconnect
Using /lib/usbmouse.o
usb.c: registered new driver usb_mouse
usbmouse.c: v1.6:USB HID Boot Protocol mouse driver
Using /lib/usbkbd.o
usb.c: registered new driver keyboard
usbkbd.c: :USB HID Boot Protocol keyboard driver
[04/Dec/2030:15:58:07 +0000] boa: server version Boa/0.94.13
[04/Dec/2030:15:58:07 +0000] boa: server built Feb 28 2004 at 21:47:23.
[04/Dec/2030:15:58:07 +0000] boa: starting server pid=34, port 80

Please press Enter to activate this console.
```

启动完毕，按照提示按下回车就可进入系统。

6.6.2 通 NFS 启动系统

在 6.1 一节中的第 7、10、11 步，您已经在主机上建立和启动了 NFS 文件系统服务。下面解释一下该服务的配置文件 exports：

/friendly-arm/root *(rw,sync,no_root_squash)

/friendly-arm/root 是代表要作为 SBC-2410X 的根文件系统的共享目录；*代表所有的客户机都可以挂接此文件系统；rw 代表客户机以读写许可来挂接它们的根文件系统；no_root_squash 选项允许客户机以主机上的 root 身份挂接根文件系统。请阅读 exports 的手册获取更详细的信息。

当按复位键启动 SBC-2410X 时，在主机的串口终端按下空格键，进入 vivi 提示符模式，输入 nfs 启动的参数(该启动参数为方便用户输入，已经作为文本文件保存在光盘的 param_nfs.txt 文件中)，该文件内容如下：

param set linux_cmd_line "console=ttyS0 root=/dev/nfs nfsroot=192.168.0.1:/friendly-arm/root ip=192.168.0.69:192.168.0.1:192.168.0.1:255.255.255.0:matrix4.arm9.net:eth0:off"

如下图所示输入 nfs 启动参数，然后使用“**param save**”命令保存：

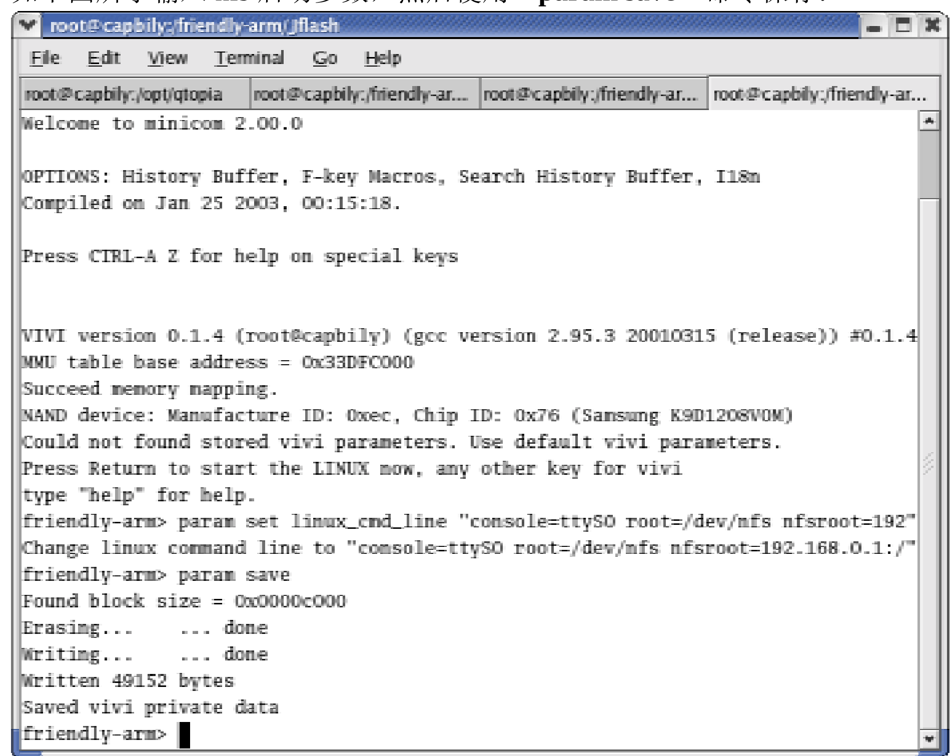


图 6-12 设置 nfs 启动参数

使用 nfs 启动的信息如下：

VIVI version 0.1.4 (root@capbily) (gcc version 2.95.3 20010315 (release)) #0.1.4 Mon Oct 4 16:20:35 CST 2004

MMU table base address = 0x33DFC000

Succeed memory mapping.

NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)

Found saved vivi parameters.

Press Return to start the LINUX now, any other key for vivi

type "help" for help.

friendly-arm>param set linux_cmd_line "console=ttyS0 root=/dev/nfs nfsroot=192.168.0.1:/friendly-arm/root ip=192.168.0.69:192.168.0.1:192.168.0.1:255.255.255.0:matrix4.arm9.net:eth0:off"

Change linux command line to "console=ttyS0 root=/dev/nfs nfsroot=192.168.0.1:/friendly-arm/root ip=192.168.0.69:192.168.0.1:192.168.0.1:255.255.255.0:matrix4.arm9.net:eth0:off"

friendly-arm> boot

Copy linux kernel from 0x00030000 to 0x30008000, size = 0x00100000 ... done

zImage magic = 0x016f2818

Setup linux parameters at 0x30000100

linux command line is: "console=ttyS0 root=/dev/nfs nfsroot=192.168.0.1:/friendly-arm/root ip=192.168.0.69:192.168.0.1:192.168.0.1:255.255.255.0:matrix4.arm9.net:eth0:off"

MACH_TYPE = 193

NOW, Booting Linux......

Uncompressing Linux... done, booting the kernel.

Linux version 2.4.18-rmk7-pxa1 (root@capbily) (gcc version 2.95.3 20010315 (release)) #9 Wed Oct 6 17:00:54 CST 2004

CPU: ARM/CIRRUS Arm920Tsid(wb) revision 0

Machine: Samsung-SMDK2410

On node 0 totalpages: 16384

zone(0): 16384 pages.

zone(1): 0 pages.

zone(2): 0 pages.

Kernel command line: console=ttyS0 root=/dev/nfs nfsroot=192.168.0.1:/friendly-arm/root ip=192.168.0.69:192.168.0.1:192.168.0.1:255.255.255.0:matrix4.arm9.net:eth0:off

DEBUG: timer count 15626

Calibrating delay loop... 99.94 BogoMIPS

Memory: 64MB = 64MB total

Memory: 62676KB available (1374K code, 364K data, 56K init)

Dentry-cache hash table entries: 8192 (order: 4, 65536 bytes)

Inode-cache hash table entries: 4096 (order: 3, 32768 bytes)
Mount-cache hash table entries: 1024 (order: 1, 8192 bytes)
Buffer-cache hash table entries: 4096 (order: 2, 16384 bytes)
Page-cache hash table entries: 16384 (order: 4, 65536 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
CPU clock = 200.000 Mhz, HCLK = 100.000 Mhz, PCLK = 50.000 Mhz
Initializing S3C2410 buffer pool for DMA workaround
usbctl: zombie --> [reset] --> default. Device in default state.
S3C2410 USB Controller Core Initialized
USB Function Character Driver Interface - 0.5, (C) 2001, Extenex Corp.
usbctl: Opened for usb-char
usbctl: Started for usb-char
Starting kswapd
devfs: v1.10 (20020120) Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x1
ttyS%d0 at I/O 0x50000000 (irq = 52) is a S3C2410
ttyS%d1 at I/O 0x50004000 (irq = 55) is a S3C2410
ttyS%d2 at I/O 0x50008000 (irq = 58) is a S3C2410
pty: 256 Unix98 ptys configured
leds initialized
S3C2410 Real Time Clock Driver v0.1
block: 128 slots per queue, batch=32
Uniform Multi-Platform E-IDE driver Revision: 6.31
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
eth0: cs8900 rev K(3.3 Volts) found at 0xd0000300
cs89x0 media RJ-45, IRQ 37
Linux video capture interface: v1.00
SCSI subsystem driver Revision: 1.00
scsi0 : SCSI host adapter emulation for IDE ATAPI devices
UDA1341 audio driver initialized
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
bon0: 00000000-00030000 (00030000) 00000000
bon1: 00030000-00130000 (00100000) 00000000
bon2: 00130000-03ff4000 (03ec4000) 00000000
usb.c: registered new driver usbdevfs

usb.c: registered new driver hub
usb-ohci.c: USB OHCI at membase 0xe9000000, IRQ 26
usb.c: new USB bus registered, assigned bus number 1
hub.c: USB hub found
port #1 suspened!
port #0 alived!
hub.c: 1 port detected
usb.c: registered new driver usblp
printer.c: v0.8:USB Printer Device Class driver
usb.c: registered new driver ov511
ov511.c: v1.48a for Linux 2.4 : OV511 USB Camera Driver
Initializing USB Mass Storage driver...
usb.c: registered new driver usb-storage
USB Mass Storage support registered.
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 4096 bind 4096)
IP-Config: Complete:
      device=eth0, addr=192.168.0.69, mask=255.255.255.0, gw=192.168.0.1,
     host=matrix4, domain=, nis-domain=arm9.net,
     bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
Looking up port of RPC 100003/2 on 192.168.0.1
Looking up port of RPC 100005/1 on 192.168.0.1
VFS: Mounted root (nfs filesystem).
Mounted devfs on /dev
Freeing init memory: 56K
==
=   Friendly-ARM Tech. Ltd.   =
=   http://www.arm9.net   =
=   http://www.arm9.com.cn   =
==
init started: BusyBox v0.60.5 (2003.09.05-09:25+0000) multi-c 齟 l binary
Using /lib/input.o
Using /lib/keybdev.o
insmod: unresolved symbol handle_scancode

```
insmod: unresolved symbol keyboard_tasklet
insmod: unresolved symbol kbd_ledfunc
Using /lib/mousedev.o
mice: PS/2 mouse device common for all mice
Using /lib/evdev.o
Using /lib/video.o
insmod: couldn't find the kernel version the module was compiled for
Using /lib/videodev.o
Linux video capture interface: v1.00
video_dev: unable to get major 81
insmod: init_module: videodev: Input/output error
Using /lib/ov511.o
usb.c: registered new driver ov511
ov511.c: v1.48a for Linux 2.4 : OV511 USB Camera Driver
Using /lib/lcd640x480x32.o
lcd address is c4960000(33800000)
LCD640x480x32 installed, by capbily@hotmail.com(http://www.arm9.net
Using /lib/hid-core.o
insmod: unresolved symbol hidinput_connect
insmod: unresolved symbol hidinput_hid_event
insmod: unresolved symbol hidinput_disconnect
Using /lib/usbmouse.o
usb.c: registered new driver usb_mouse
usbmouse.c: v1.6:USB HID Boot Protocol mouse driver
Using /lib/usbkbd.o
usb.c: registered new driver keyboard
usbkbd.c: :USB HID Boot Protocol keyboard driver
[04/Dec/2030:15:57:15 +0000] boa: server version Boa/0.94.13
[04/Dec/2030:15:57:15 +0000] boa: server built Feb 28 2004 at 21:47:23.
[04/Dec/2030:15:57:15 +0000] boa: starting server pid=37, port 80

Please press Enter to activate this console.



BusyBox v0.60.5 (2003.09.05-09:25+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

sh: can't access tty; job control turned off
```

[root@fa /]#

6.6 更新系统

更新系统包括：

- 更新 bootloader(vivi)
- 更新内核 kernel
- 更新文件系统

下面主要介绍了在各种情况下如何更新系统。

6.6.1 系统完整的时候如何重新系统

不管是旧版本的 cramfs 还是新版本的 yaffs，只要系统能正常通过网络文件系统 NFS 启动，就可以使用本节介绍的方法进行更新。

首先使用网络文件系统(nfs)作为根目录启动系统，按回车进入系统后执行“bk”命令，如下图所示。

注：bk 是一个脚本文件，该脚本首先使用 imagewrite 对 flash 进行分区，然后烧写 bootloader 和 kernel。关于 imaewrite 的使用方法可以参考其帮助信息。

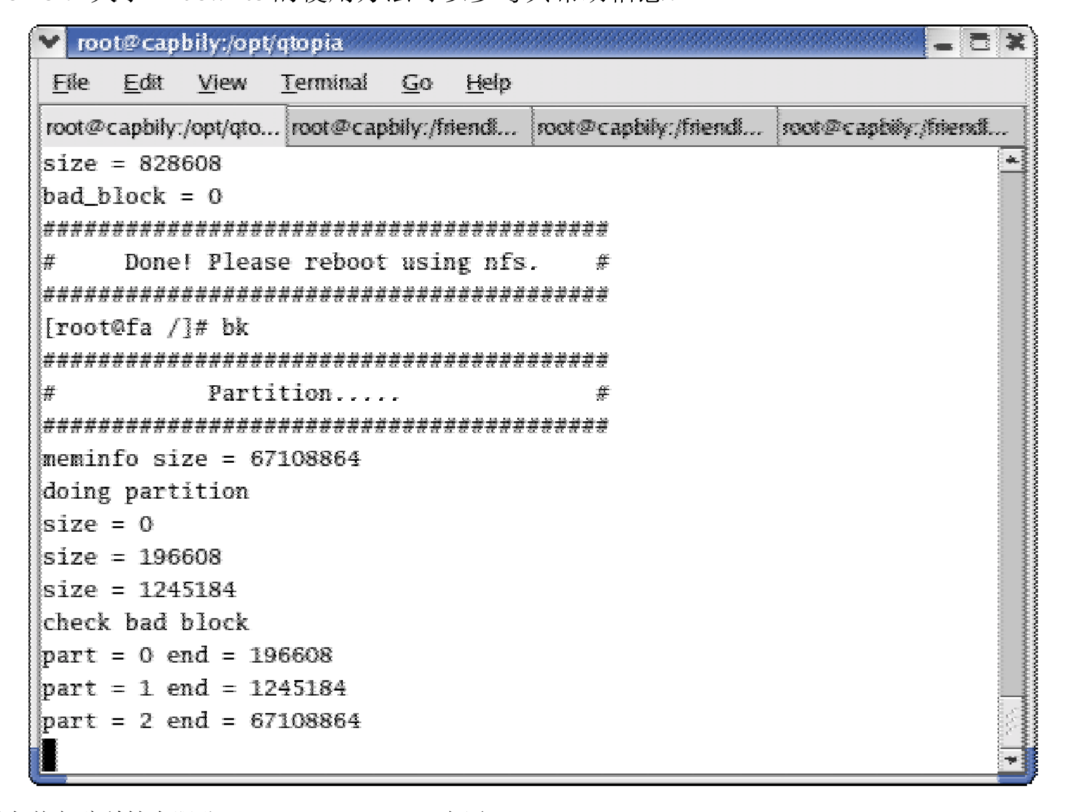


图 **6-13** 执行 **bk** 开始分区

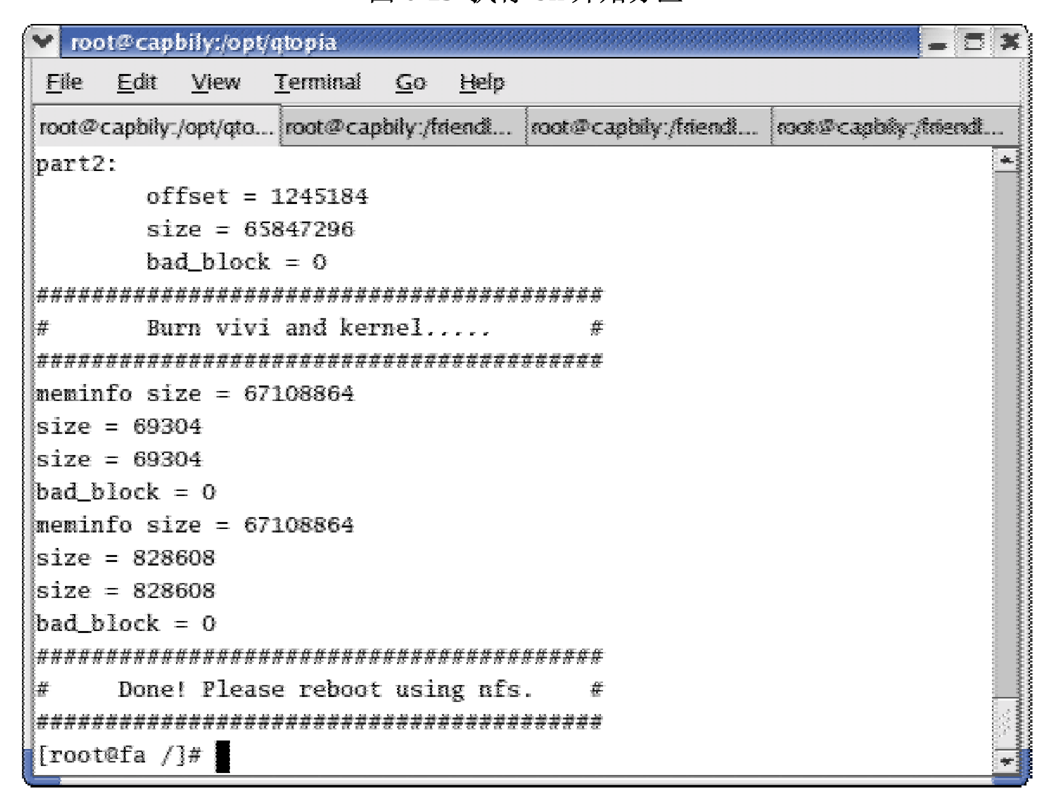


图 **6-14 bk** 执行完毕

bk 执行完毕，按照提示，再次使用 nfs 启动系统，注意，此时旧版本的 vivi 提示信息已经从“**vivi>**”变为“**friendly-arm>**”。

然后使用“bs”命令先对 flash 进行格式化，再安装根文件系统内容，如下图所示。

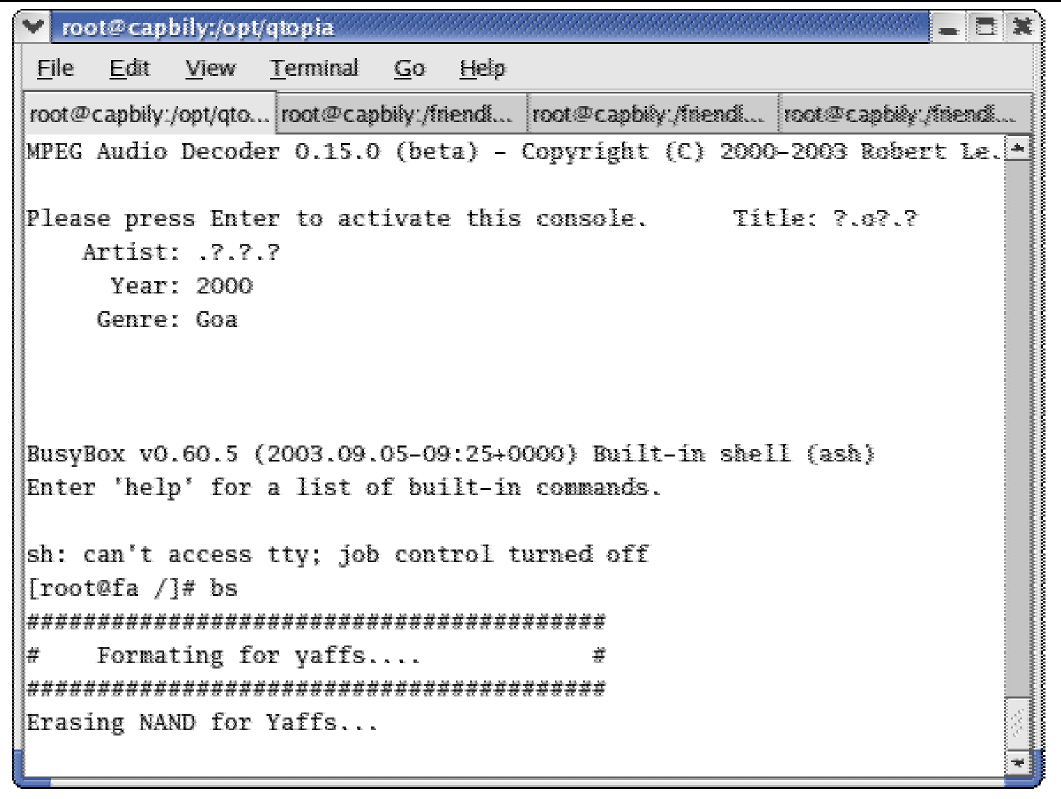


图 6-15 格式化 flash

当出现下图所示的信息时，系统更新完毕，用户就可以使用新系统了。

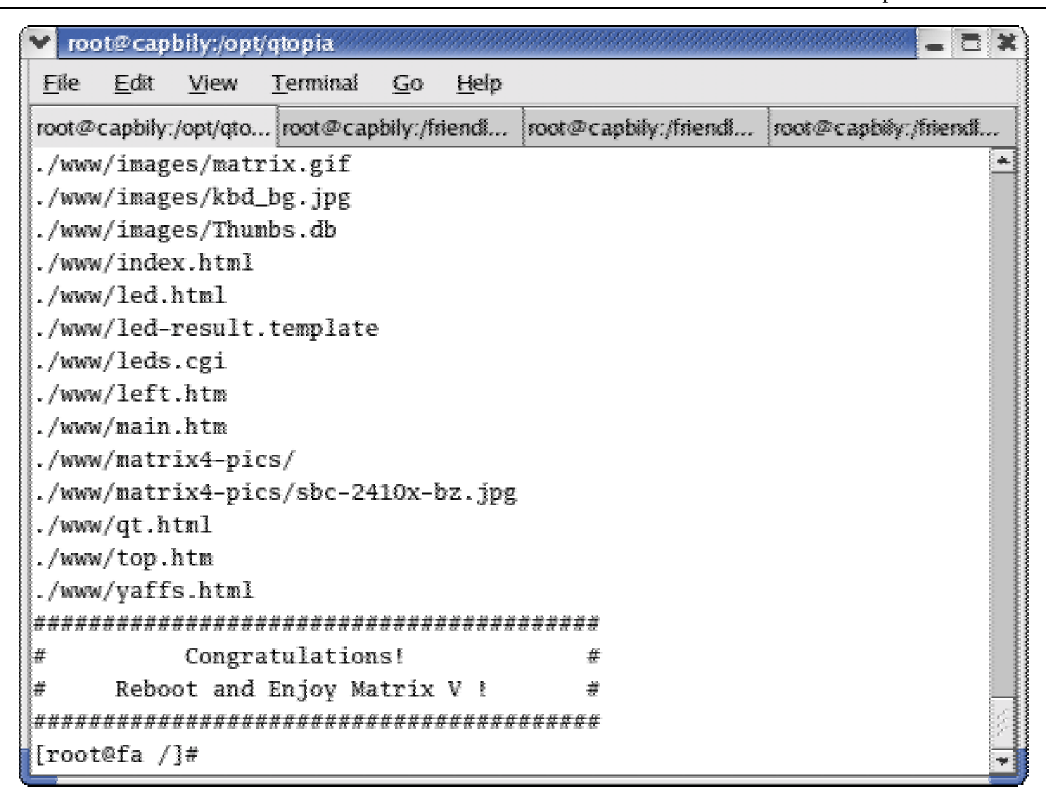


图 6-16 系统更新完毕

6.6.2 系统崩溃后如何重新系统（高级用法）

当整个系统崩溃或者是全新的时候，使用该方法更新系统。

注意：在对整个系统操作不熟悉的情况下，谨防破坏系统。

(1)使用 JTAG 接口下载 vivi

使用 JTAG 接口下载程序需要一条 JTAG 电缆和一个 Jflash 烧写程序。把 JTAG 电缆连接 SBC-2410X 和主机的并口，然后打开目标板电源开关

Jflash 及其用法在/friendly-arm/Jflash 目录下，进入该目录：

#cd /friendly-arm/Jflash

执行以下命令可以查看 Jflash 的使用方法：

#./Jflash –s3c2410--help

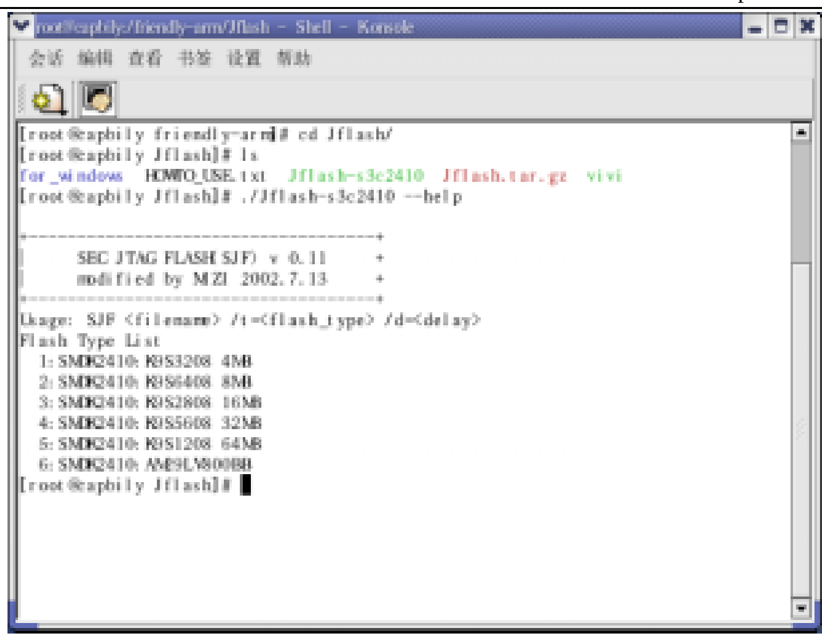


图 6-17 JTAG 的使用方法

可见，对应不同类型的 Flash ，Jflash 程序使用不同的选项参数，因为我们使用的是 64M 三星 Nand Flash，因此使用“/t=5”。

执行以下命令开始烧写 vivi，如图所示。

#./Jflash-s3c2410 vivi /t=5

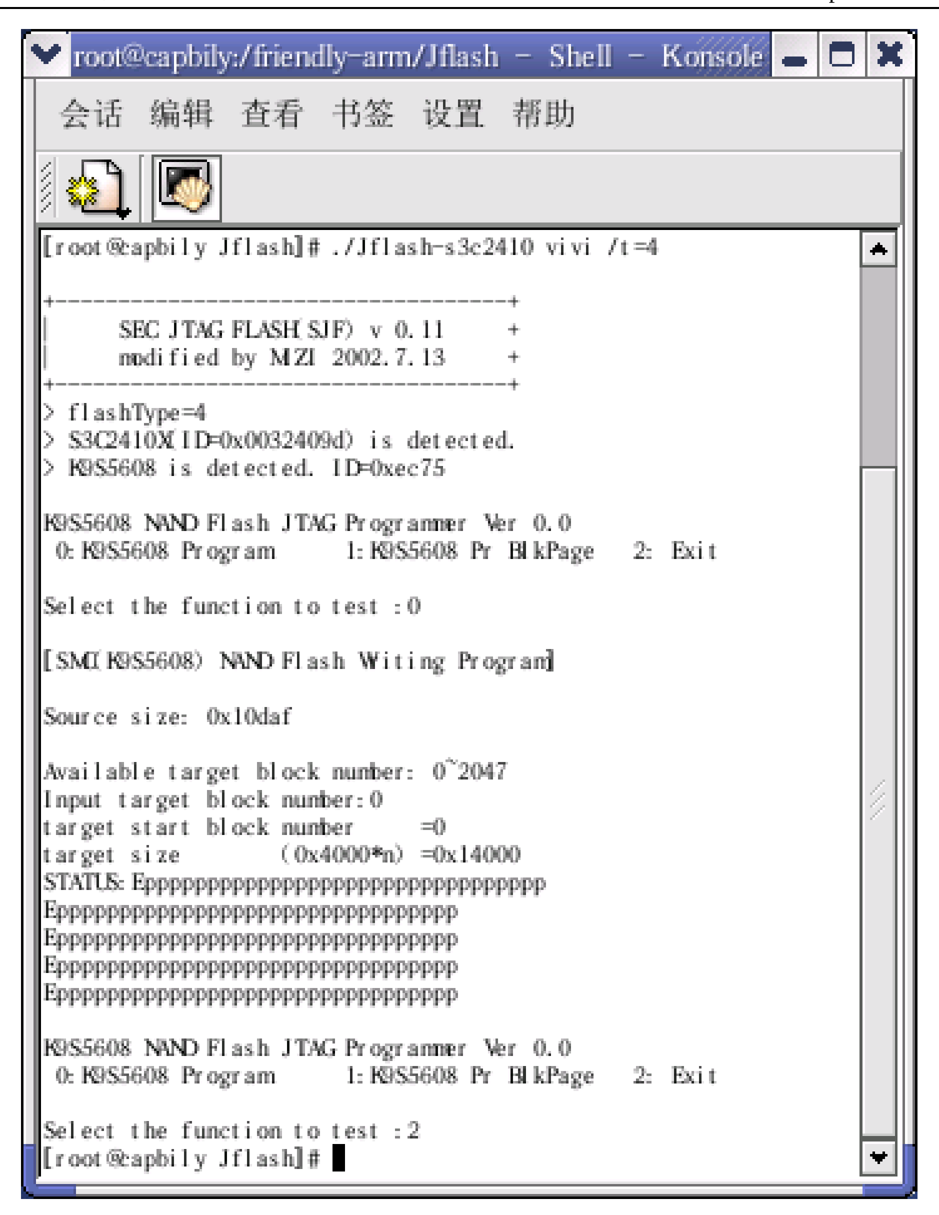


图 6-18 使用 JTAG 烧写 vivi

当 vivi 烧入 Flash，并能正常启动，就可以使用它来下载内核了，vivi 的详细使用方法可以参考附录 B“**VIVI 的使用方法**”，下面我们介绍如何使用 vivi 下载内核。

连接好串口，运行 minicom 终端程序，因为 SBC-2410X 附带的 vivi 启动等待时间很短，因此请按下空格键，然后打开电源开关，就会进入 vivi 的命令提示符下，如图所示。

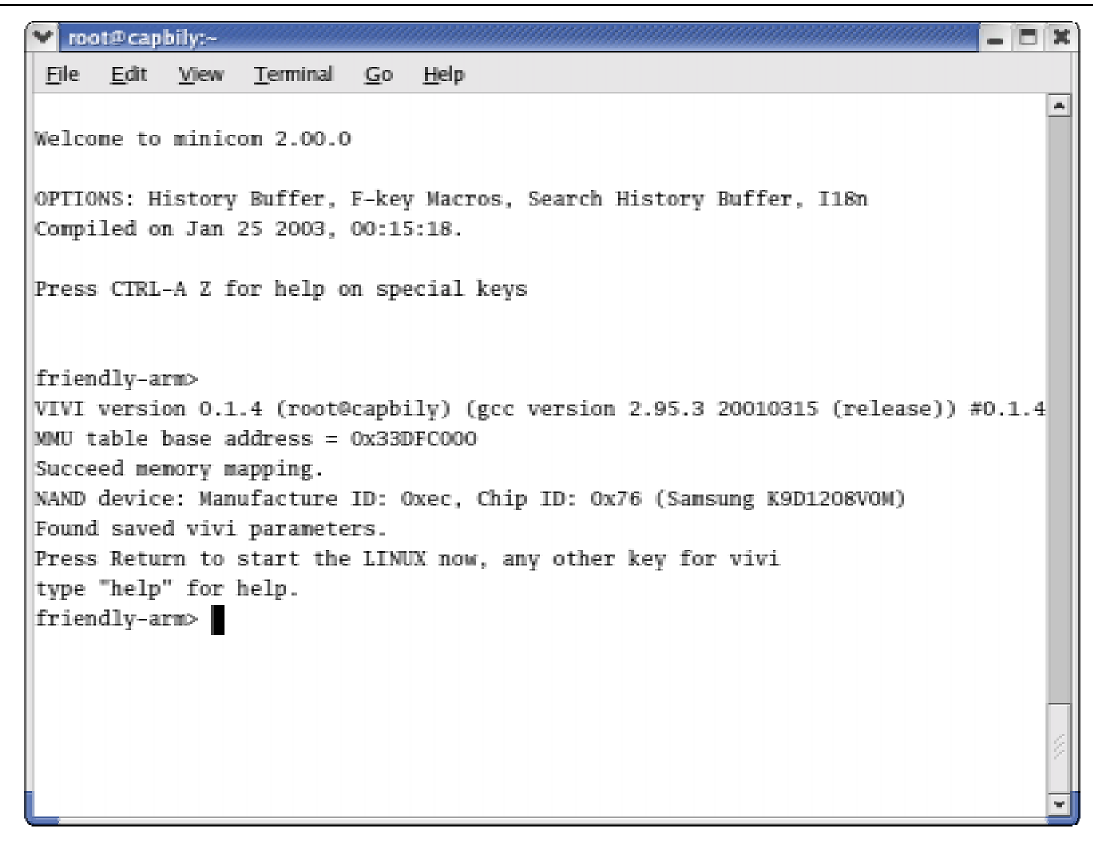


图 **6-19** 按空格键进入 **vivi** 的命令模式

执行“**load flash kernel x**”命令，开始下载内核，如图 7-16 所示，该命令的解释如下：

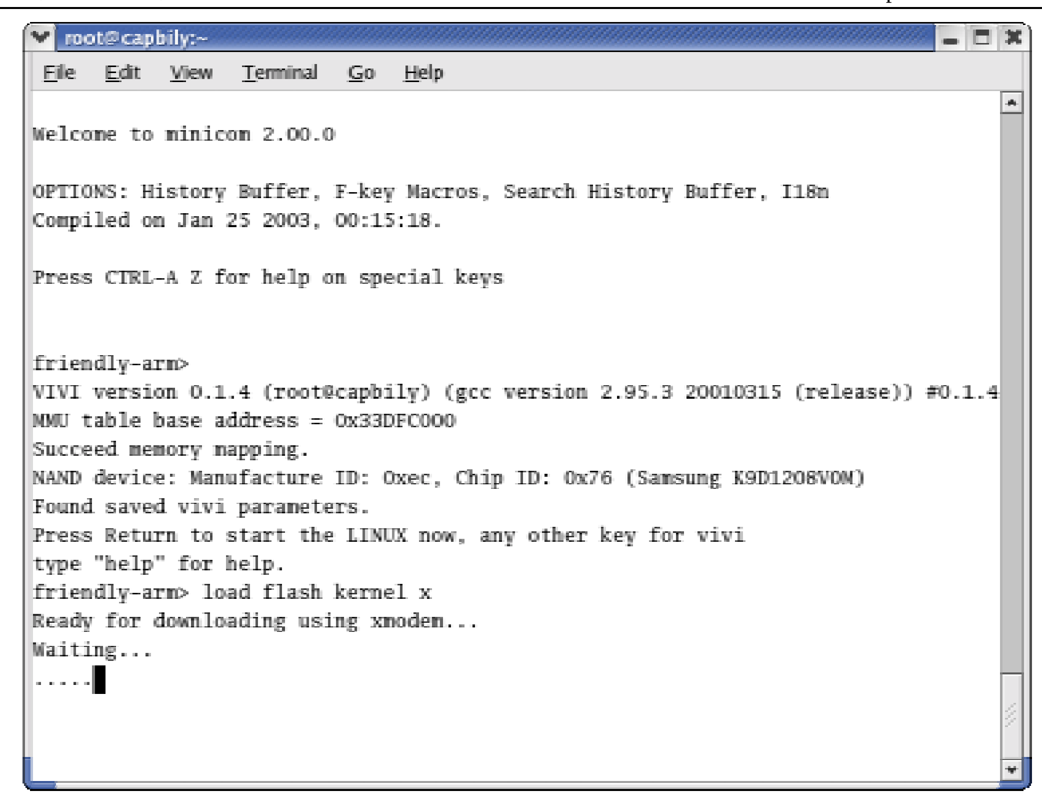
load – vivi 的下载命令

flash – 把文件下载到 flash 中

kernel – 要下载的文件是 kernel 类型，和分区参数同名

x – 使用超级终端的 xmdoem 协议下载。

此时先按下“Ctrl”，不要松开，再按下“a”键，然后同时松开，再按下“s”键，进入下载模式，如图所示选择 xmodem 协议方式下载。



```
root@capbily:~
File Edit View Terminal Go Help

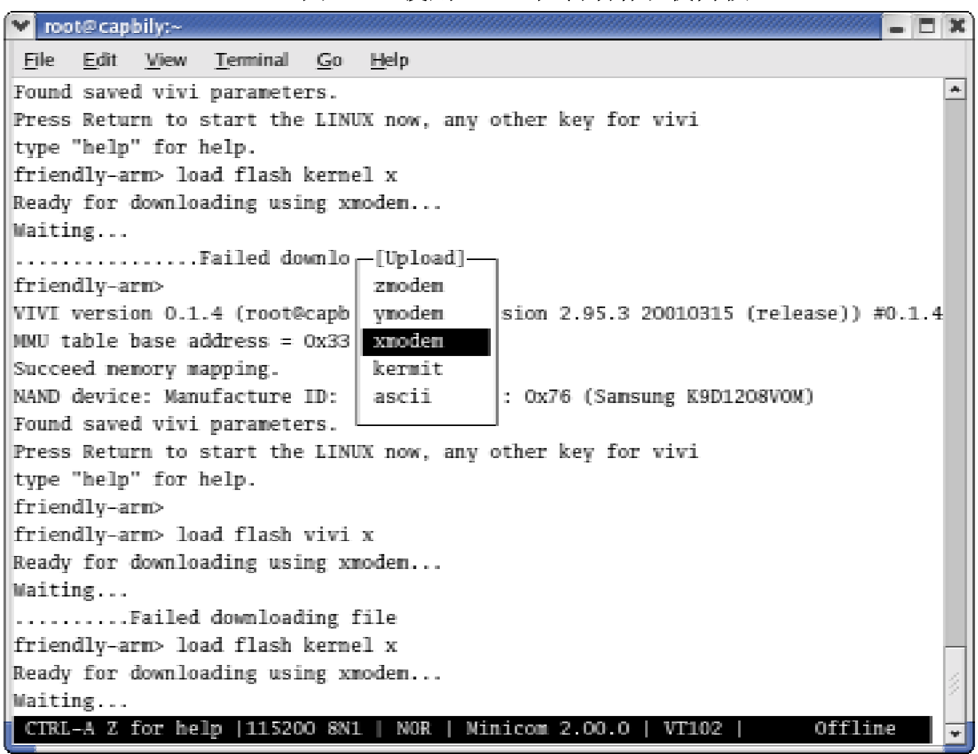
Welcome to minicom 2.00.0

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n
Compiled on Jan 25 2003, 00:15:18.

Press CTRL-A Z for help on special keys


friendly-arm>
VIVI version 0.1.4 (root@capbily) (gcc version 2.95.3 20010315 (release)) #0.1.4
MMU table base address = 0x33DFC000
Succeed memory mapping.
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Found saved vivi parameters.
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
friendly-arm> load flash kernel x
Ready for downloading using xmodem...
Waiting...
.....
```

图 6-20 使用 **load** 命令开始下载内核



```
root@capbily:~
File Edit View Terminal Go Help
Found saved vivi parameters.
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
friendly-arm> load flash kernel x
Ready for downloading using xmodem...
Waiting...
................Failed downlo ┌[Upload]─┐
friendly-arm>                │ zmodem  │
VIVI version 0.1.4 (root@capb│ ymodem  │sion 2.95.3 20010315 (release)) #0.1.4
MMU table base address = 0x33│ xmodem  │
Succeed memory mapping.      │ kermit  │
NAND device: Manufacture ID: │ ascii   │: 0x76 (Samsung K9D1208V0M)
Found saved vivi parameters. └─────────┘
Press Return to start the LINUX now, any other key for vivi
type "help" for help.
friendly-arm>
friendly-arm> load flash vivi x
Ready for downloading using xmodem...
Waiting...
..........Failed downloading file
friendly-arm> load flash kernel x
Ready for downloading using xmodem...
Waiting...
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
```

图 6-21 选择 xmodem 协议

按回车，再按回车输入要下载的文件，注意最好把要下载的文件复制到/root 目录。如图 6-22 所示，图 6-23 为下载过程。

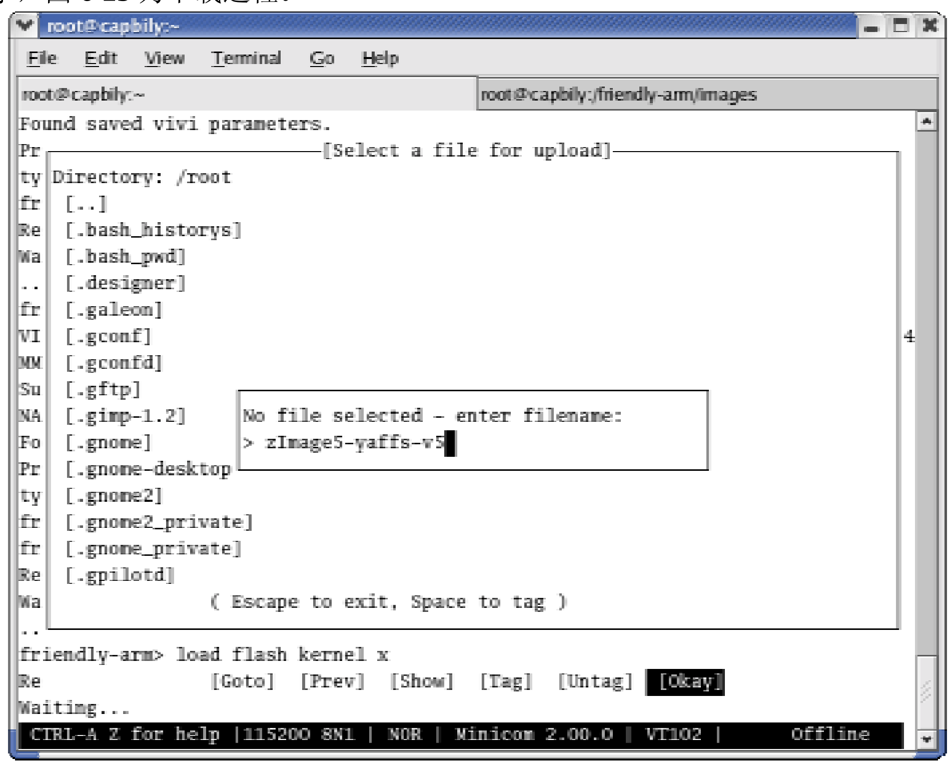


图 6-22 选择要发送的文件和并选择 Xmodem 协议发送

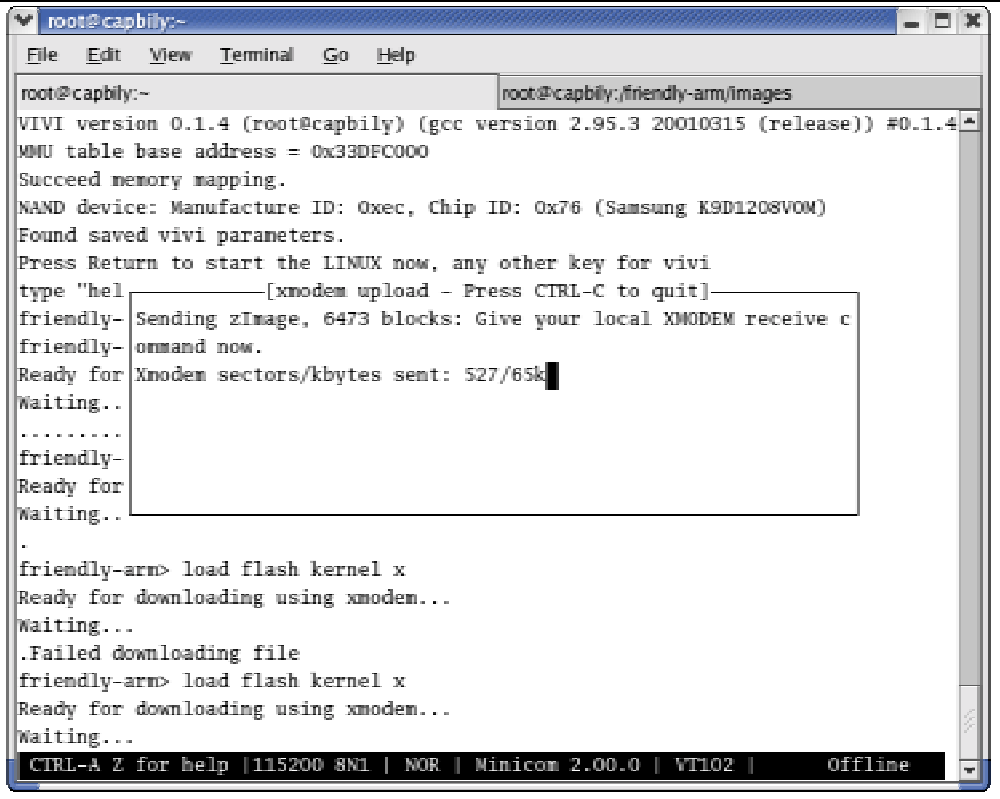


图 6-23 kernel 下载过程

发送文件结束，vivi 将自动保存所下载的文件到 flash 中，如图 6-24 所示。

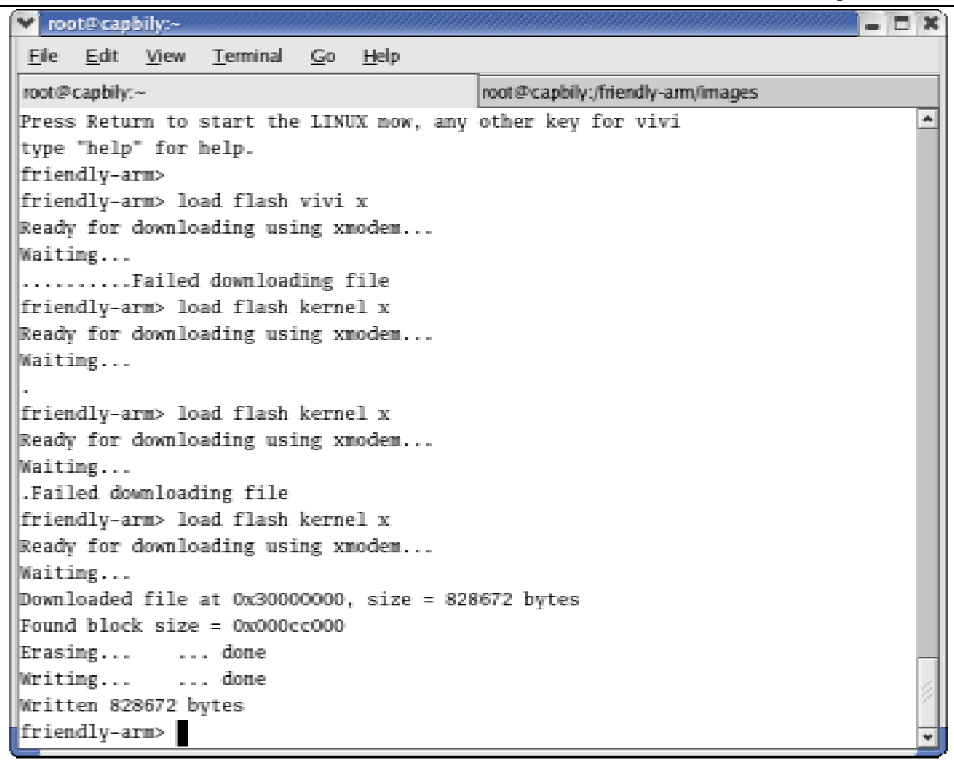


图 6-24 vivi 自动把内核保存到 flash 中

此时，如果输入“**boot**”命令，vivi 将会启动刚刚下载的内核，但是无法进入系统，因为还没有文件系统，此时用户可以按照“系统的启动方式”一节选择 NFS 启动系统，然后按照 6.6.1 介绍的方法，使用 NFS 更新系统。

第七节 嵌入式用户图形界面编程

7.1 嵌入式图形系统简介

7.1.1 Qt/Embedded

Qt/Embedded(简称 QtE)是一个专门为嵌入式系统设计图形用户界面的工具包。Qt 是挪威 Trolltech 软件公司的产品，它为各种系统提供图形用户界面的工具包，QtE 就是 Qt 的嵌入式版本。

使用 QtE，开发者可以：

- 用 QtE 开发的应用程序要移植到不同平台时，只需要重新编译代码，而不需要对代码进行修改。
- 可以随意设置程序界面的外观。
- 可以方便地为程序连接数据库
- 可以使程序本地化
- 可以将程序与 Java 集成

嵌入式系统地要求是小而快速，而 QtE 就能帮助开发者为满足这些要求开发强壮地应用程序。QtE 是模块化和可裁剪地。开发者可以选取他所需要的一些特性，而裁剪掉所不需要的。这样，通过选择所需要的特性，QtE 的映像变得很小，最小只有 600K 左右。

同 Qt 一样，QtE 也是用 C++写的，虽然这样会增加系统资源消耗，但是却为开发者提供了清洗的程序框架，使开发者能够迅速上手，并且能够方便地编写自定义的用户界面程序。由于 QtE 是作为一种产品推出，所以它有很好的开发团体和技术支持，这对于使用 QtE 的开发者来说，方便开发过程，并增加了产品的可靠性。

总的来说，QtE 拥有下面一些特征：

- 拥有同 Qt 一样的 API；开发者只需要了解 Qt 的 API，不用关心程序所用到的系统与平台
- 它的结构很好地优化了内存和资源地利用。
- 拥有自己的窗口系统：QtE 不需要一些子图形系统。它可以直接对底层的图形驱动进行操作。
- 模块化：开发者可以根据需要自己定制所需要的模块。
- 代码公开以及拥有十分详细的技术文档帮助开发者
- 强大的开发工具
- 与硬件平台无关：QtE 可以应用在所有主流平台和 CPU 上。支持所有主流的嵌入式

Linux，对于在 Linux 上的 QtE 的基本要求只不过是 Frame Buffer 设备和一个 C++编译器(如 gcc)。Qte 同时也支持很多实时的嵌入式系统，如 QNX 和 WindowsCE。

- 提供压缩字体格式：即使在很小的内存中，也可以提供一流的字体支持。
- 支持多种的硬件和软件的输入。
- 支持 Unicode，可以轻松地使程序支持多种语言。
- 支持反锯齿文本和 Alpha 混合的图片。

Trolltech 公司在 QtE 的基础上开发了一个应用的环境—Qtopia，这个应用环境为移动和手持设备开发。其特点就是拥有完全的、美观的 GUI，同时它也提供可上百个应用程序用于管理用户信息、办公、娱乐、Internet 交流等。已经有很多公司采用了 Qtopia 来开发他们主流的 PDA。

QtE 虽然公开代码和技术文档，但是它不是免费的，当开发者的商业化产品需要用到他的运行库时，必须向 Trolltech 公司支持 license 费用(每套 3 美金)，如果开发的东西不用于商业用途则不需要付费。QtE 由于平台无关性和提供了很好的 Gui 编程接口，在许多嵌入式系统中得到了广泛的应用，是一个成功的嵌入式 GUI 产品。

7.1.2 Microwindows/Nano-X

Mirowindows 是嵌入式系统中广为使用的一种图形用户接口，其官方网站是：http://www.microwindows.org。这个项目的早期目标是在嵌入式 Linux 平台上提供和普通个人电脑上类似的图形用户界面。作为 PC 上 X-Windows 的替代品，Microwindows 提供了和 X-Windows 类似的功能，但是占用的内存要少得多，根据用户得配置，Microwindows 占用得内存资源在 100KB-60KB。Microwindows 支持多种外部设备得输入，包括液晶显示器、鼠标和键盘等。在嵌入式 Linux 平台上，从 Linux2.2.x 的内核开始，为了方便图形的显示，使用了 framebuffer 的技术。Microwindows 完全支持 Linux 最新 framebuffer 技术，支持每个象素 1 位、2 位、4 位、8 位、16 位、24 位和 32 位的色彩空间/灰度，并且通过调色板技术将 RGB 格式的颜色空间转换成目标机器上最相近的颜色，然后显示出来。

Microwindows 的核心基于显示设备接口，因此可移植行很好，microwindows 有自己的 Framebuffer，因此它并不局限于 Linux 开发平台，在 eCos、FreeBSD、RTEMS 等操作系统上都能很好地运行。

此外，Microwindows 能在宿主机上仿真目标机。这意味着基于 Linux 的 Microwindows 应用程序的开发和调试可以在普通的个人电脑上进行，而不需要使用普通嵌入式软件的“宿主机－目标机”调试模式，从而大大加快了开发速度。

Microwindows 起源于 NanoGUI 项目，早期 Microwindows 有两个版本，一个版本包含了一组和微软的 WIN32 图形用户接口相似的 API，这个版本就是 Microwindows 版本；另外一个版本是基于 X-Windows 的一组 Xlib 风格的 API 函数库，这个版本允许 X11 的二进制代码直接在 Microwindows 的 Nanx-X 服务器上运行，称之为 Nano-X。目前 Microwindows 的最新版本是 2003 年 5 月底发布底 0.90 版本。

Mincrowindows 是完全免费的一个用户图形系统。

7.1.3 MiniGUI

MiniGUI 是由北京飞漫软件技术有限公司主持的一个自由软件项目(遵循 GPL 条款)，其目标是为基于 Linux 的实时嵌入式系统提供一个轻量级的图形用户界面支持系统。

MiniGUI 为应用程序定义了一组轻量级的窗口和图形设备接口。利用这些接口，每个应用程序可以建立多个窗口，而且可以在这些窗口中绘制图形。用户也可以利用 MiniGUI 建立菜单、按钮、列表框等常见的 GUI 元素。

用户可以将 MiniGUI 配置成“MiniGUI-Threads”或者“MiniGUI-Lite。运行在 MiniGUI-Threads 上的程序可以在不同的线程中建立多个窗口，但所有的窗口在一个进程中运行。相反，运行在 MiniGUI-Lite 上的每个程序是单独的进程，每个进程也可以建立多个窗口。MiniGUI-Threads 适合于具有单一功能的实时系统，而 MiniGUI-Lite 则适合于类似于 PDA 和瘦客户机等嵌入式系统。

如果用户的专用商业产品使用了 MiniGUI，需要向飞漫公司支持商业授权费用。

7.2 基于 Qt/Embedded 的嵌入式 GUI 设计

本小节内容主要介绍如何把 QtE 应用程序程序移植到 SBC-2410X，关于 QtE 详细的编程可以参考 QtE 电子开发文档，它位于光盘的 Documents 目录，来自解压开的 Qtopia/docs 目录。

7.2.1 建立 Qt/Embedded 开发环境

Qt 开发环境的建立十分简单。

把光盘放入 DVD-ROM，执行以下步骤：

Step1 挂接光盘

#mount /dev/cdrom /mnt/cdrom

Step2 进入 Qt 开发包目录

#cd /mnt/cdrom/SBC-2410X-Linux/EmGUI

Step3 安装 X86 版本的 Qt 和 Qtopia 源代码

#tar xvzf x86-qtopia.tgz –C /friendly-arm

Step4 安装 SBC-2410X 版本的 Qtopia 源代码

#tar xvzf arm-qtopia.tgz –C /friendly-arm

Step5 弹出光盘

#cd /

#eject

当在 PC 上模拟 Qtopia 的运行时，需要用到对应 Qt 版本的库文件，因此需要修改 **/etc/ld.so.conf** 文件以适应刚刚安装的 Qt(Redhat 安装时带有 Qt 库，但不适合我们最新安装的版本)，修改后的 ld.so.conf 文件内容如下：

/friendly-arm/x86-qtopia/qt/lib

/friendly-arm/x86-qtopia/qtopia/lib

/usr/kerberos/lib

/usr/X11R6/lib

/usr/lib/sane

/usr/lib/mysql

修改完此文档后，为了让刚刚安装的库生效，必须运行 ldconfig。

至此 Qt 的开发缓建已经建立。

7.2.2 基于 PC 的 Hello, SBC-2410X

(1) 编译 Qt 及 Qtopia

#cd /friendly-arm/x86-arm

#./build ；在本地目录执行编译命令，此后将设置好 QT 运行环境

(2) 编译 Hello

#cd hello

#make

hello 编译完以后会直接生成到$QPEDIR/bin 目录下

(3) 运行 Hello

为了让 helllo 在 qptoia 中运行，需要一个配置文件 hello.desktop，拷贝该文件到 qtopia 的应用程序配置目录

#cp hello.desktop $QPEDIR/apps/Applications/

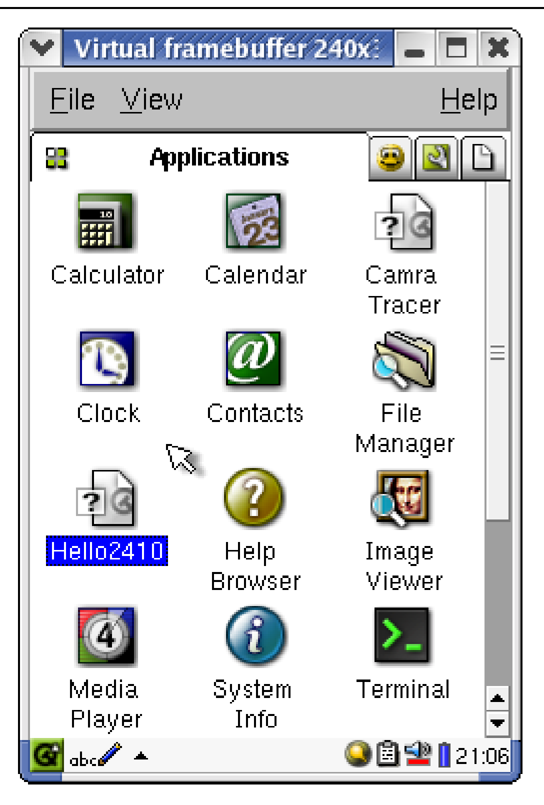
然后运行虚拟 Framebuffer

#qvfb & ；因为 QT 环境变量已经设置好，所以直接执行

最后执行 qpe

#qpe ；因为 QT 环境变量已经设置好，所以直接执行

如图 7-1 所示，在 qpotia 中出现 hello 程序：



Hello2410 在 Qtopia 中

点击运行 hello2410，则出现图 7-2 所示的 Hello 程序：



图 7-2 Hello is Running

如果以后重新登录，则需要重新设置 QT 环境变量，请运行以下脚本命令：

#cd /friendly-arm/x86-arm

#. set-env ；注意：先敲一个“.”，然后一个空格，再输入“set-env”

hello

7.2.3 移植到 SBC-2410X

把 hello 程序移植到 SBC-2410X，仅仅是修改了 hello 目录下的 Makefile，源代码不用作任何修改。

(1)编译 Qt 及 Qtopia for ARM

#cd /friendly-arm/arm-arm

#./build ；在本地目录执行编译命令，并设置好 QT for ARM 编译环境

(2)编译 hello

#cd hello

#make

hello 将生成在/friendly-arm/arm/qtopia/bin 目录下

(3)在 SBC-2410X 上运行 hello

把生成的 hello 二进制可执行文件和 hello.desktop 分别复制到 SBC-2410X 系统中的/opt/qtopia/bin 目录和/opt/qtopia/ apps/Applications/目录

重新启动整个系统或者单独重启 Qtopia 系统，将会看到 hello 程序已经存在，点击运行将会出现和上面一样的结果。

7.3 基于 Microwindows/Nano-X 的嵌入式 GUI 设计

(待续)

7.3.1 建立 Microwindows/Nano-X 开发环境

7.3.2 基于 PC 的 Hello，SBC-2410X

7.3.3 移植到 SBC-2410X

附录 A：Windows 超级终端图解

为了通过串口连接SBC-2410X，必须使用一个模拟终端程序，几乎所有的类似软件都可以使用，其中MS-Windows 自带的超级终端是最常用的选择，当你安装Windows9x 时需要自定义选择安装该项，Windows2000 及更高版本则已经缺省安装；在Linux 操作系统下minicom 是一个非常好用的虚拟终端程序，大多数发行版安装时均已包含了该程序，使用时你只要设置一下即可，其他.类似软件在这里就不介绍了。

在此首先着重介绍一下Windows 自带的超级终端程序并以Windows2000 为例，或许其他Windows 版本的程序界面有所不同。

超级终端程序通常位于附件中的通讯中，它是一个文件夹点开始按钮直至找到该项点击打开该目录如图4-2 所示



图4-2 超级终端文件夹

"Hypertrm" 图标所指即是超级终端程序，当该程序第一次运行时，会跳出图4-3所示窗口，询问你是否要安装一个Modem，此时你不需要安装Modem，因此点“否”按钮。

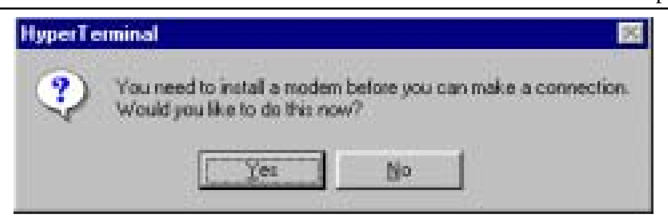


图 **4-3** 超级终端询问对话框，点“**No**”

接下来，超级终端会要求你为新的连接取一个名字，如图 4-4 所示，这里我取了”ttyS0”，Windows 系统会禁止你取类似”COM1”这样的名字，因为这个名字被系统占用了。



图 **4-4** 为新的连接命名

当你命名完以后，又会跳出一个对话框，你需要选择连接 SBC-2410X 的串口，我这里选择了串口 1，如图 4-5 所示：

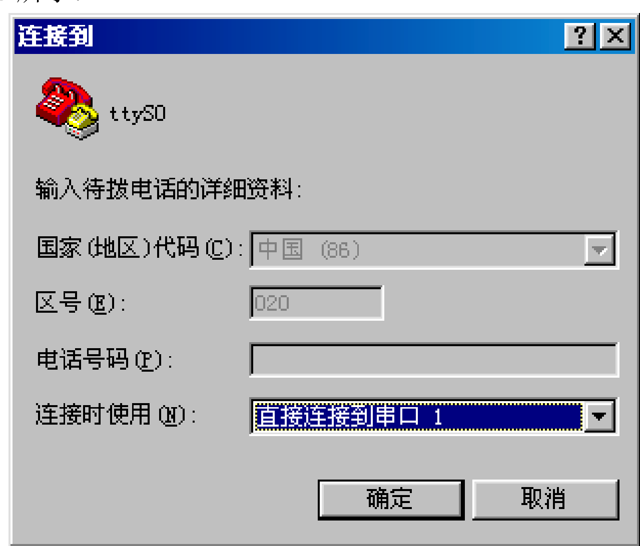


图 **4-5** 选择和实验板相连的串口

最后，最重要的一步是设置串口，注意必须选择无流控制，否则，或许你只能看到输出而不能输入，另外 SBC-2410X 工作时的串口波特率是 115200，如图 4-6 所示。



图 4-6 设置串口通信参数

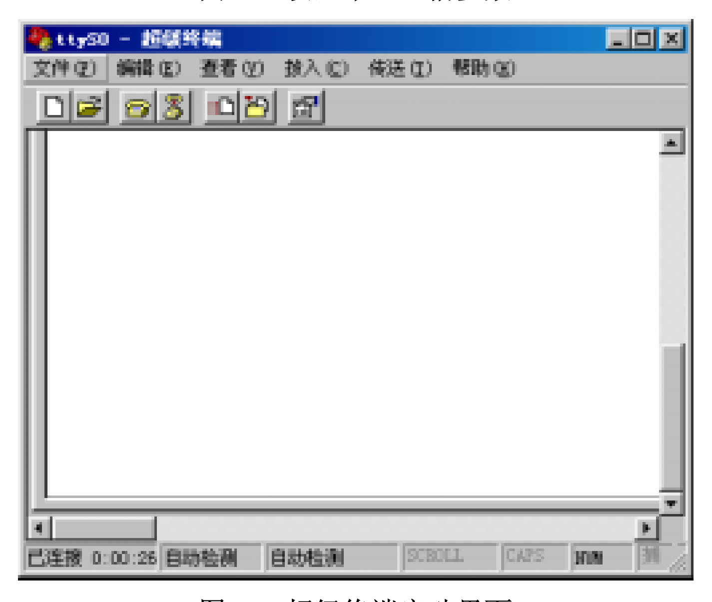


图 4-7 超级终端启动界面

当所有的连接参数都设置好以后，超级终端会显示入图 4-7 所示主窗口，此时串口是空白的，打开电源开关，系统会出现如图 4-7 所示 vivi 启动界面。

按下回车键启动 arm-linux 操作系统，如图 4-8 所示为 arm-linux 启动界面。此时再根据屏

幕提示按“回车”键即可进入 Linux 系统。

选择超级终端“文件”菜单下的“另存为…”，保存该连接设置，以便于以后再连接时就不必重新执行以上设置了。

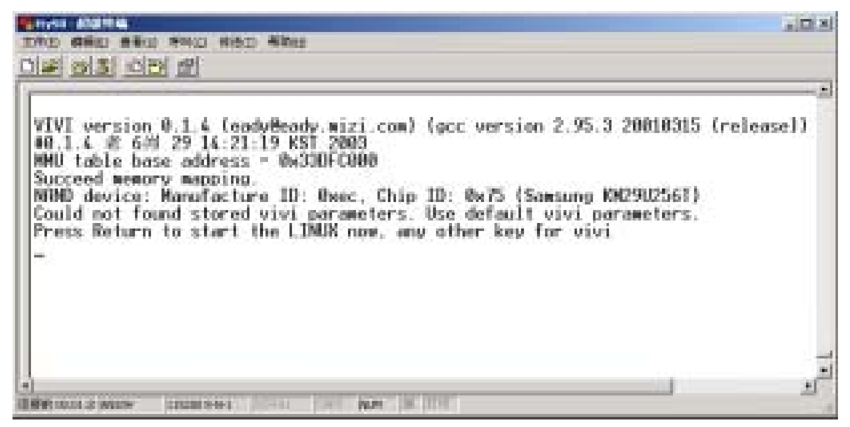
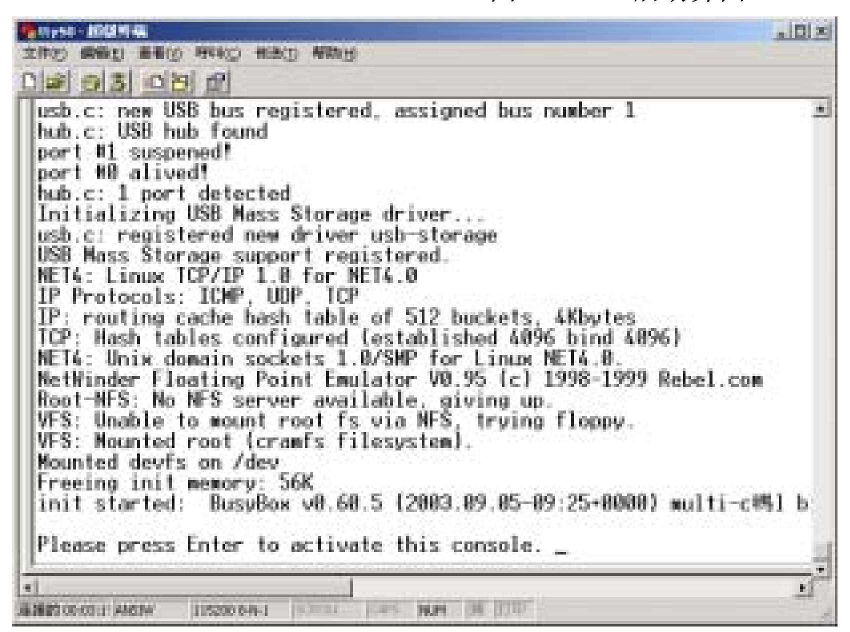


图 4-7 **VIVi** 启动界面

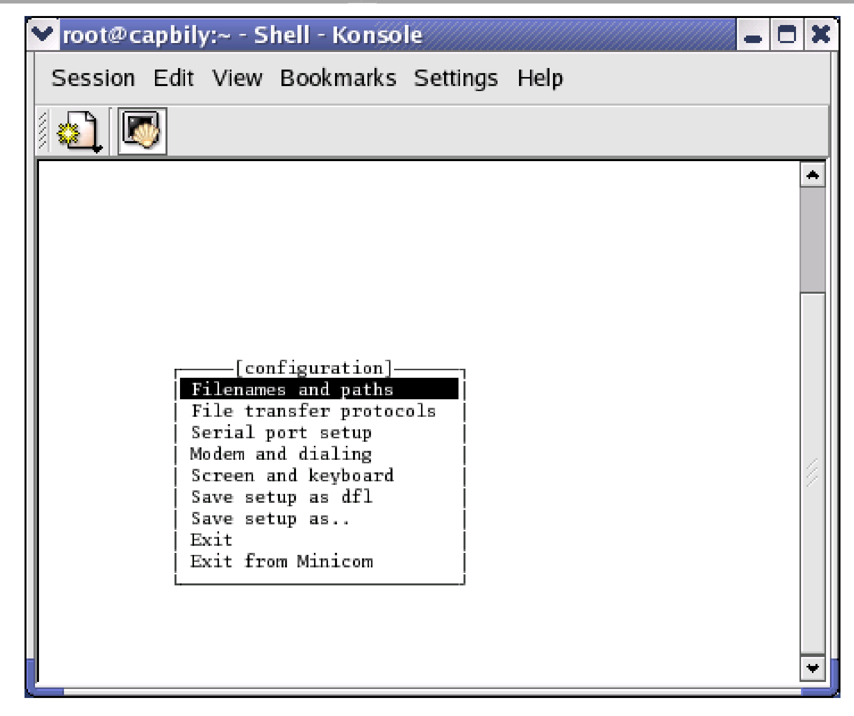


附录 B：minicom 的使用方法

minicom 是 Linux 下类似“超级终端”的程序，一般完全安装大部分发行版的 Linux 时都会包含它，下面介绍它的使用方法。

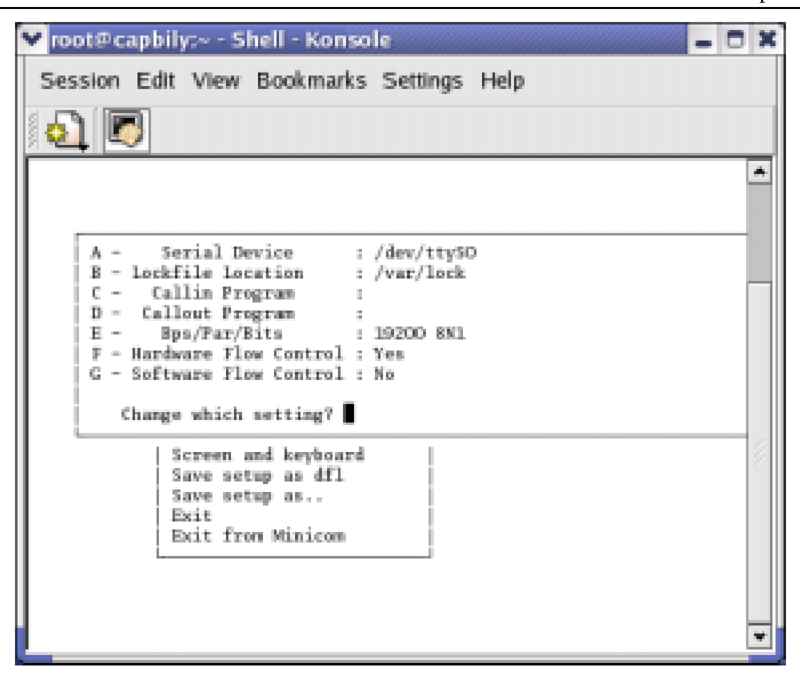
使用 minicom 之前先设置一下，如下图所示：

```
#minicom -s                                        ；加“-s”选项设置 minicom
```



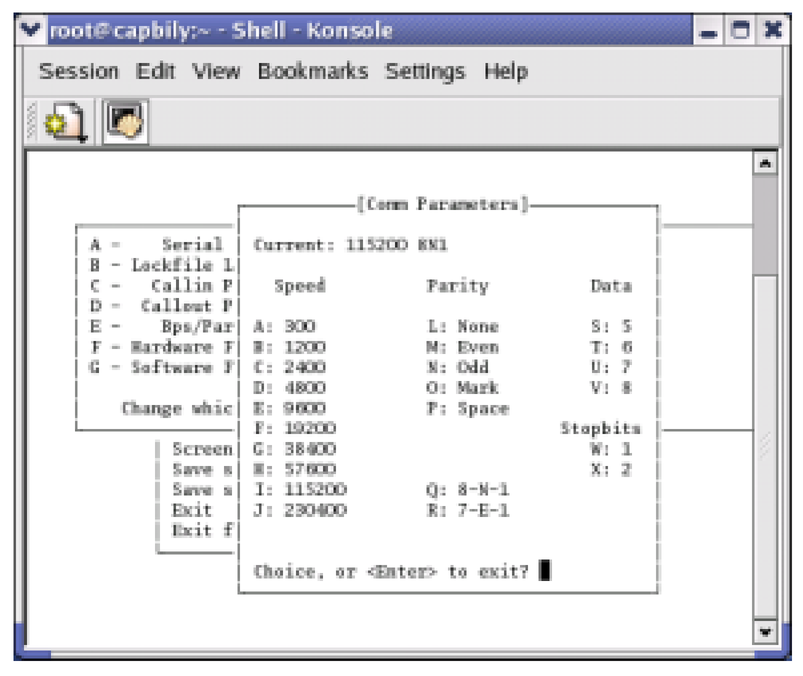
运行“minicom -s”设置 minicom

选择菜单中的“Serial port setup”，按回车，进入如下图所示界面。此时按“A”以设置“Serial Device”（如果您使用串口 1，则输入/dev/ttyS0，如果您使用串口 2，则输入/dev/ttyS1）。



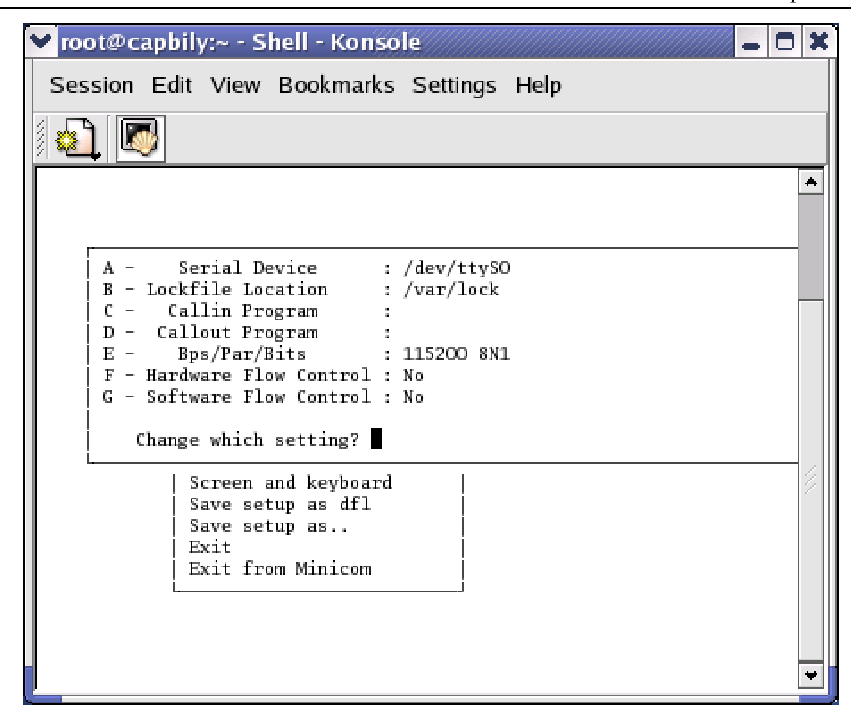
设置“**Serial Device**”

按“E”键进入设置“bps/par/Bits”（波特率）界面，如下图所示。再按“I”以设置波特率为 115200。



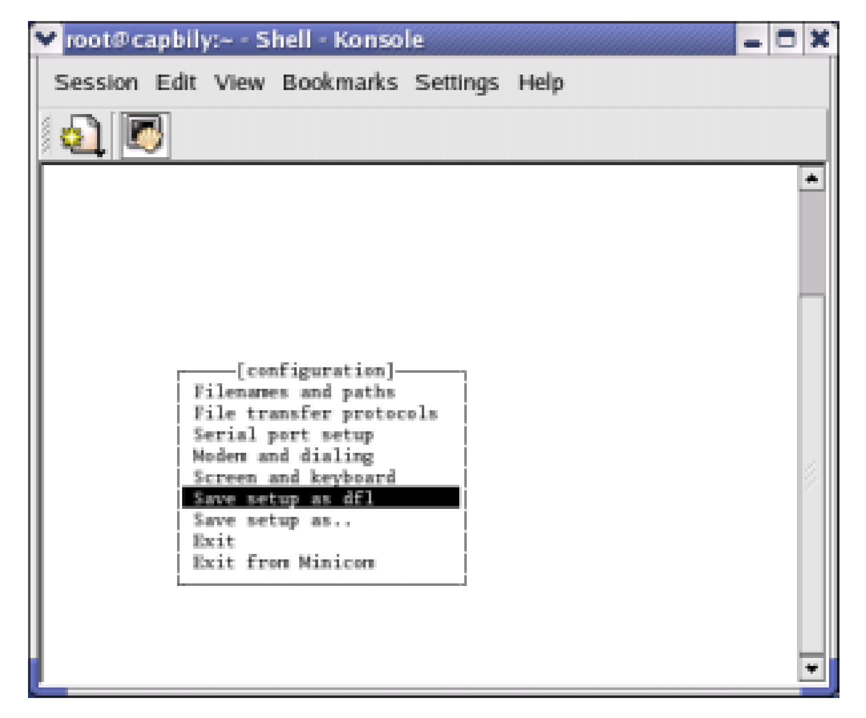
设置波特率

然后按回车退回到上一级菜单，按“F”键设置“Hardware Flow Control”为“NO”，其他选项使用缺省值，如下图所示。



设置“**Hardware Flow Control**”

设置完毕，按回车键返回到串口设置主菜单，选择“Save setup as dfl”，按回车键保存刚才的设置，如下图所示。

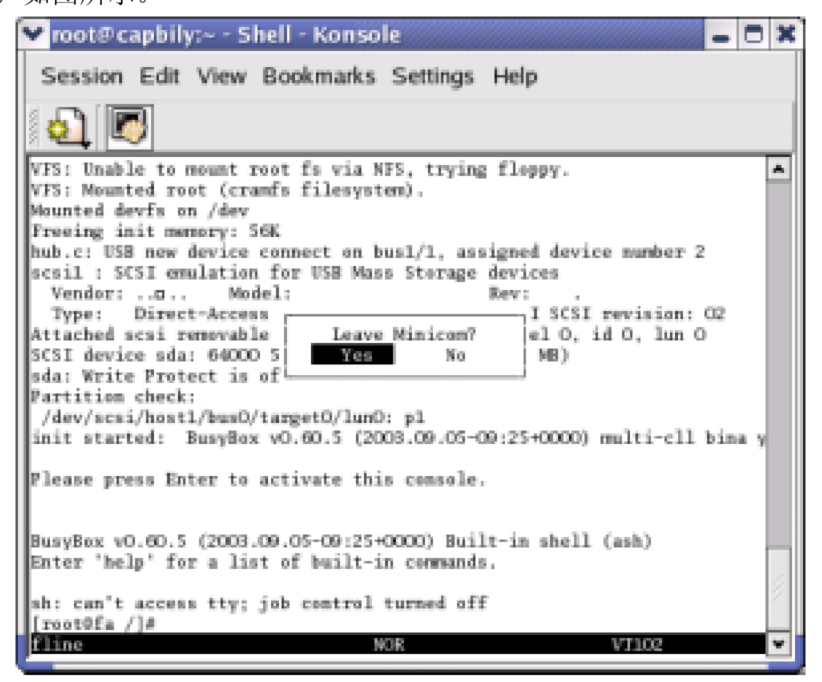


保存设置

再选择“Exit”退出设置模式。刚才的设置被保存到“/etc/minirc.dfl”。

设置完毕，如果此时打开 SBC-2410X 电源的电源开关，就会看到 vivi 的启动信息，当 Linux 启动以后，minicom 就等于相当于虚拟终端，你就能通过它来操作目标板了。

要退出 minicom，同时按下“Ctrl+A”键，送开后紧接着再按下“Q”键，在跳出的窗口中，选择“Yes”，如图所示。



退出 minicom

附录 C：VIVI 的使用方法

本文档取自 VIVI 的 guide.txt（稍作编辑）

Getting started with vivi

Janghoon Lyu (nandy@mizi.com)
This is a short introduction about the vivi.
Revision History
Revision v0.1 2002-10-21 Revised by: jl
Initial public release. very very short descriptions. (with foolish sentences -_-;;)

Table of Contents

----

1. Introduction

This document explains to you:

1) compiling the vivi
2) interface an user and the vivi
3) avaliable user commands

1.1. Definitions

vivi

The vivi is one of arm boot loaders. Autoboot mode The vivi has two modes. one of these is autoboot mode. This is a default mode. In this mode, The vivi automatically boot-up the linux kernel when a delay time is expired.

Prompt mode

One of vivi's modes is prompt mode. In this mode, An user is able to command to the vivi doing something.

Host platform

Generally, an engineer develop the software on the powerful destktop computer (not on the embedded machines). This descktop called host platform.

Target platform

Target platform means developemnt boards or embedded machines.

2. Compiling the vivi

This section describes how to compile the vivi.

2.1. Pre-Requirements

Here is a list of the minimum levels of software necessary to compile the vivi.

* Gnu C 2.95.3
* Gnu make 3.79.1
* binutils 2.11.2

And then, your host platform are installed these tools.

* You have a cross-compiler and tool-chains.
* You have a Baurne-Again shell. (/bin/bash)
* You have a ncurses library.

2.2. Straight-forward compilation

The compiliation(configuration) environment of the vivi is simlilar to the environment of the linux kernel. So, you can do make distclean, make clean, make oldconfig , make menuconfig, and make config. General compiling sequence sequence is:

1) make distclean
2) make menuconfig
3) make

Note that If you change configurations, run make clean and make. Next two sections describes compilation of two examples.

2.3. SA-1110-Based machines with the NOR flash

I will take an example about the vivi compliation. I assumed that a target platform's core block is composed like this:

* CPU: SA-1110 processor
* ROM: 32MB Intel Strata Flash (32-bit wide)
* RAM: 32MB DRAM

These features make sense to you. Many platforms I have met is designed core block like above. For instance, KINGS, GILL, and ENDA's core block is designed like this. Therefore, you refer to vivi/arch/def-configs/kings, vivi/arch/def-configs/gill, and vivi/arch/d
ef-configs/enda

If you want to compile vivi for KINGS(i.e. a configuration of a target board, alredy exist in the vivi/arch/def-cofnigs), just do like this:

make kings

Other machines can complie same ways.

2.4. S3C2410-Based machines with the NAND flash

Here, I will take an example about a S3C2410-Based machine. The SMDK-2410 (a development board by SAMSUNG) is a S3C2410-Based machine.The core block of SMDK-2410 is composed like this:

* CPU: S3C2410
* ROM: 64MB SMC (NAND Flash)
* RAM: 64MB DRAM

A configuration for SMDK-2401 will be found atvivi/arch/def-configs/smdk2410. To compile for SMDK-2410, do like this(As I described above):

make smdk2410

3. Using the vivi

3.1. Interface between an user and the vivi

The vivi uses the serical communication for user interface. Therefore,

to connect the vivi, you 1) connect a serial cable between host platform and target platform. 2) already have a serial communication program like the minicom. 3) properly make the vivi binary to support a UART port when you configure the vivi.

If all of above is ok, you can see messages on the screen printed by the vivi. For example, below messages catched from the SMDK-2410

```
VIVI version 0.1.4 (nandy@nandy.mizi.com) (gcc version 2.95.2 20000516 (release)
[Rebel.com]) #0.1.4 荐 10 岿 16 16:19:11 KST 2002
MMU table base address = 0x33DFC000
Succeed memory mapping.
NAND device: Manufacture ID: 0xec, Chip ID: 0x75 (Samsung KM29U256T)
Found saved vivi parameters.
Press Return to start the LINUX now, any other key for vivi
```

See the last line on the screen. (As I mentioned section 1.1) the vivi has two mode: an autoboot mode and a prompt mode. The vivi wait for a key input. If an user want to enter the prompt mode, press andy key (except Enter key). And then you can see the "vivi>" prompt. Otherwise, the vivi try boot the linux kernel after waiting a few seconds or minutes.

3.2. Built-in user commands

This is not full-described built-in user comamnd. But following commands is enough to use the vivi (as far as I know).

3.2.1. load command

A load command is loading binaries to the flash or the ram.

Usage:

load <media_type> [<partname> | <addr> <size>] <x|y|z>

<media_type>

This argument is where to load. Availabe values are *flash* and *ram*.

[<partname>] or [<addr> <size>]

This arguement determines location where to load a binary. If you want to use pre-defined mtd partiton informations, just type a partition name. Otherwise you specify an address and a size.

<x|y|z>

This arguement determines the file transfer protocol. I shy that the vivi only supprot xmodem curruntly. So, Available value is "x".

For exampe, you load zImage to flash memroy.

vivi > load flash kernel x

or you specify an address and a size.

vivi > load flash 0x80000 0xc0000 x

3.2.2. part command

The vivi has mtd partiton informations for the vivi. This informatin not related to mtd partition informations of mtd device drivers. The vivi uses partition informations when load a binary, boot the linux kernel, erase flash memroy, etc...

Avalilable commands are:

Display mtd partition informations.

part show

Add a new mtd partition.

part add <name> <offset> <size> <flag>

<name> is name of a new mtd partiton.

<offset> is offset in the mtd device.

<size> is a size of a mtd parition.

<flag> is flags of a mtd parition. Available valuse are JFFS2, LOCKED, and BONFS.

Delete a mtd partiton.

part del <partname>

Reset mtd partitions to default values.

part reset

Save paramter valuse and mtd parition informations to flash permanently.

part save

3.2.3. param command

The vivi has some parameter values. For example, the "boot_delay" paramter determines how long wait keystroek when the vivi is in the autoboot mode. I'm sorry that this feature is in progress. So all listed parameter is not available.

Here, I give several tips for you.

If you change the "linxu command line",

vivi> param set linux_cmd_line "you wish.."

If you want to see paramters,

vivi> param show

If you want to wait a long time when recevie file via xmodem,

vivi> param set xmodem_initial_timeout 3000000

If you want to boot imediately when a hardware reset,

vivi> param set boot_delay 100000

3.2.4. boot command

A boot command is boot the linux kernel which is stored in the flashmemroy or ram.

Usage:

boot <media_type> [<partname> | <addr> <size>]

<media_type>

This argument is where to store the linux kernel image. Availabe values are ram, nor and smc.

[<partname>] or [<addr> <size>]

This arguement determines location where to store the linux kernel. If you want to use pre-defined mtd partiton information, just type a partition name. Otherwise you shuld specify an address and a size.

Note that all argument is optional. If you omit all argument (just type boot), all arguments is parsing from pre-defined mtd parititon
information called "kernel" For example,

vivi> boot

the vivi read the linux kernel binary from the "kernel" mtd parititons.

vivi> boot nor 0x80000

the vivi read the linux kernel binary form the nor flash memory. The offset is 0x80000 and the size is default value (0xc0000). Ocassionally, you want to test the kernel

on the ram (not store kernel
to flash). you can do it on the vivi On the SA-1110 based machine,

vivi> load ram 0xc0008000 x
vivi> boot ram

On the S3C2410 based machine,

vivi> load ram 0x30008000 x
vivi> boot ram

On the PXA-240 based machine,

vivi> load ram 0xA0008000 x
vivi> boot ram

the vivi boot the linux kernel from the ram.

3.2.5. flash command

A flash comamnd manages the flash memory. If you want to erase flash memory,

flash erase [<partname> | <offset> <size>]